

---

# Amazon SageMaker

## Developer Guide



---

## Amazon SageMaker: Developer Guide

## Table of Contents

What Is Amazon SageMaker? .....	1
Amazon SageMaker Features .....	1
Amazon SageMaker Pricing .....	3
Are You a First-time User of Amazon SageMaker? .....	3
How It Works .....	3
Machine Learning with Amazon SageMaker .....	4
Explore, Analyze, and Process Data .....	5
Fairness and Model Explainability .....	6
Best Practices for Evaluating Fairness and Explainability in the ML Lifecycle .....	7
Sample Notebooks .....	7
Guide to the SageMaker Clarify Documentation .....	8
Model Training .....	8
Model Deployment .....	10
Hosting Services .....	10
Batch Transform .....	13
Validating Models .....	14
Model Monitoring .....	15
ML Frameworks and Toolkits .....	16
Apache MXNet .....	16
Apache Spark .....	17
Chainer .....	25
Hugging Face .....	26
PyTorch .....	27
R .....	28
Scikit-learn .....	30
SparkML Serving .....	31
TensorFlow .....	31
Supported Regions and Quotas .....	32
Request a service quota increase for SageMaker resources .....	32
Get Started .....	34
Set Up Amazon SageMaker .....	34
Create an Amazon Account .....	34
Create an IAM Administrator User and Group .....	34
Onboard to Studio .....	35
Onboard Using Quick Start .....	35
Onboard Using IAM .....	36
Choose a VPC .....	38
Delete a Domain .....	39
SageMaker JumpStart .....	40
Using JumpStart .....	43
Solutions .....	44
Models .....	44
Deploy a model .....	44
Model Deployment Configuration .....	44
Fine-Tune a Model .....	46
Fine-Tuning Data Source .....	46
Fine-Tuning deployment configuration .....	49
Hyperparameters .....	49
Training Output .....	50
Next Steps .....	50
Studio Tour .....	51
Get Started with Notebook Instances .....	51
Machine Learning with the SageMaker Python SDK .....	52
Tutorial Overview .....	52

Step 1: Create an Amazon SageMaker Notebook Instance .....	53
Step 2: Create a Jupyter Notebook .....	54
Step 3: Download, Explore, and Transform Data .....	55
Step 4: Train a Model .....	59
Step 5: Deploy the Model .....	63
Step 6: Evaluate the Model .....	65
Step 7: Clean Up .....	68
SageMaker Studio .....	69
UI Overview .....	69
Left sidebar .....	70
File and resource browser .....	71
Main work area .....	72
Settings .....	72
Use the Studio Launcher .....	72
Notebooks and compute resources .....	73
Utilities and files .....	73
Studio Entity Status .....	74
Use Studio Notebooks .....	76
Compare Studio Notebooks with Notebook Instances .....	77
Get Started .....	78
Create or Open a Notebook .....	79
Use the Toolbar .....	80
Share and Use a Notebook .....	82
Get Notebook and App Metadata .....	84
Get Notebook Differences .....	85
Manage Resources .....	86
Usage Metering .....	92
Available Resources .....	92
Bring your own image .....	95
Create an image .....	96
Attach an image .....	97
Launch a custom image .....	98
Bring your own image tutorial .....	101
Custom image specifications .....	109
Set Up a Connection to an EMR Cluster .....	110
Perform Common Tasks .....	113
Upload Files .....	114
Clone a Git Repository .....	114
Stop a Training Job .....	114
Use TensorBoard in Amazon SageMaker Studio .....	115
Manage Your EFS Volume .....	116
Provide Feedback .....	117
Update Studio and Apps .....	117
Studio Pricing .....	119
Troubleshooting .....	119
Notebook Instances .....	120
Create a Notebook Instance .....	120
Access Notebook Instances .....	122
Update a Notebook Instance .....	123
Customize a Notebook Instance .....	124
Lifecycle Configuration Best Practices .....	125
Install External Libraries and Kernels in Notebook Instances .....	126
Notebook Instance Software Updates .....	128
Control an Amazon EMR Spark Instance Using a Notebook .....	129
Example Notebooks .....	131
Use or View Example Notebooks in Jupyter Classic .....	131
Use or View Example Notebooks in Jupyterlab .....	131

Set the Notebook Kernel .....	132
Git Repos .....	133
Add a Git Repository to Your Amazon SageMaker Account .....	134
Create a Notebook Instance with an Associated Git Repository .....	136
Associate a CodeCommit Repository in a Different Amazon Account with a Notebook Instance ...	137
Use Git Repositories in a Notebook Instance .....	138
Notebook Instance Metadata .....	140
Monitor Jupyter Logs in Amazon CloudWatch Logs .....	140
Autopilot: Automated ML .....	142
Get started .....	143
Samples .....	143
Videos .....	144
Tutorials .....	145
Create an Autopilot experiment .....	145
Problem types .....	148
Regression .....	149
Binary classification .....	149
Multiclass classification .....	149
Model support and validation .....	149
Autopilot algorithm support .....	149
Autopilot cross-validation .....	150
Deploy models .....	150
Explainability .....	152
Models generated .....	153
Notebooks generated .....	156
Data exploration notebook .....	157
Candidate definition notebook .....	157
Configure inference output .....	157
Inference container definitions for regression and classification problem types .....	157
Select inference response for classification models .....	158
Quotas .....	159
Quotas that you can increase .....	159
Resource quotas .....	160
API reference .....	161
SageMaker API reference .....	161
Amazon SageMaker Python SDK .....	162
Amazon Command Line Interface (CLI) .....	162
Amazon SDK for Python (Boto) .....	162
Amazon SDK for .NET .....	163
Amazon SDK for C++ .....	163
Amazon SDK for Go .....	163
Amazon SDK for Java .....	163
Amazon SDK for JavaScript .....	163
Amazon SDK for PHP V3 .....	164
Amazon SDK for Ruby V3 .....	164
Label Data .....	165
Label Data (Ground Truth) .....	165
Are You a First-time User of Ground Truth? .....	166
Getting started .....	166
Label Images .....	171
Label Text .....	188
Label Videos and Video Frames .....	197
Label 3D Point Clouds .....	230
Verify and Adjust Labels .....	293
Creating Custom Labeling Workflows .....	300
Create a Labeling Job .....	332
Use Input and Output Data .....	362

Enhanced Data Labeling .....	427
Security and Permissions .....	439
Monitor Labeling Job Status .....	454
Create and Manage Workforces .....	456
Using the Amazon Mechanical Turk Workforce .....	457
Managing Vendor Workforces .....	460
Use a Private Workforce .....	461
Crowd HTML Elements Reference .....	482
SageMaker Crowd HTML Elements .....	482
Augmented AI Crowd HTML Elements .....	553
Prepare and Analyze Datasets .....	561
Detect Pretraining Data Bias .....	561
Amazon SageMaker Clarify Terms for Bias and Fairness .....	561
Sample Notebooks .....	562
Measure Pretraining Bias .....	563
Generate Reports for Bias in Pretraining Data in SageMaker Studio .....	572
Prepare Data with Data Wrangler .....	577
Get Started with Data Wrangler .....	578
Import .....	588
Create and Use a Data Wrangler Flow .....	618
Transform Data .....	621
Analyze and Visualize .....	638
Export .....	645
Shut Down Data Wrangler .....	650
Update Data Wrangler .....	581
Security and Permissions .....	653
Release Notes .....	661
Troubleshoot .....	662
Process Data .....	664
Sample Notebooks .....	664
CloudWatch Logs and Metrics .....	665
Data Processing with Apache Spark .....	665
Running a Spark Processing Job .....	665
Data Processing with scikit-learn .....	666
Use Your Own Processing Code .....	666
Run Scripts with a Processing Container .....	667
Build Your Own Processing Container .....	668
Create, Store, and Share Features .....	673
How Feature Store Works .....	673
Create Feature Groups .....	674
Find, Discover, and Share Features .....	674
Real-Time Inference for Features Stored in the Online Store .....	674
Offline Store for Model Training and Batch Inference .....	675
Feature Data Ingestion .....	675
..... .....	675
Get started with Amazon SageMaker Feature Store .....	675
Feature Store Concepts .....	675
Create Feature Groups .....	676
Adding required policies to your IAM role .....	686
Use Feature Store with Studio .....	691
Data Sources and Ingestion .....	698
Stream Ingestion .....	699
Data Wrangler with Feature Store .....	699
Athena and Amazon Glue with Feature Store .....	701
Security and Access Control .....	702
Using KMS Permissions for Amazon SageMaker Feature Store .....	702
Authorizing Use of Your CMK for Your Online Store .....	703

Using Grants to Authorize Feature Store .....	705
Monitoring Feature Store interaction with Amazon KMS .....	705
Accessing Data in Your Online Store .....	705
Authorizing Use of Your CMK for Offline Store .....	705
Cross-Account Offline Store Access .....	705
Step 1: Set Up the Offline Store Access Role in Account A .....	706
Step 2: Set up an Offline Store S3 Bucket in Account B .....	707
Step 3: Set up an Offline Store KMS Encryption Key in Account A .....	707
Step 4: Create a Feature Group in Account A .....	709
Quotas, Naming Rules and Data Types .....	709
Limits and Quotas .....	709
Naming Rules .....	709
Data Types .....	709
Amazon SageMaker Feature Store Offline Store Data Format .....	710
Amazon SageMaker Feature Store Notebook Examples .....	710
Feature Store sample notebooks .....	711
Training .....	713
Choose an algorithm .....	713
Choose an algorithm implementation .....	714
Problem types for the basic machine learning paradigms .....	716
Use built-in algorithms .....	718
Use Reinforcement Learning .....	1552
Experiments .....	1553
SageMaker Experiments Features .....	1553
Create an Experiment .....	1554
View and Compare Experiments, Trials, and Trial Components .....	1558
Track and Compare Tutorial .....	1561
Search Experiments Using Studio .....	1567
Clean Up Experiment Resources .....	1572
Search Using the Console and API .....	1574
Debugger .....	1579
Debugger Features .....	1579
Supported Frameworks and Algorithms .....	1581
Debugger Architecture .....	1582
Tutorials .....	1584
Configure Debugger Using SageMaker Python SDK .....	1593
Configure Debugger Using SageMaker API .....	1615
List of Built-in Rules .....	1626
Create Custom Rules .....	1670
Use Debugger with Custom Training Containers .....	1672
Action on Debugger Rules .....	1675
Debugger on Studio .....	1685
Debugger Interactive Reports .....	1700
Analyze Data Using the SMDebug Client Library .....	1724
Visualize Debugger Output Tensors in TensorBoard .....	1732
Best Practices for Debugger .....	1734
Advanced Topics and Reference .....	1737
Automatic Model Tuning .....	1745
How Hyperparameter Tuning Works .....	1745
Define Metrics .....	1747
Define Hyperparameter Ranges .....	1748
Tune Multiple Algorithms .....	1749
Example: Hyperparameter Tuning Job .....	1754
Stop Training Jobs Early .....	1764
Run a Warm Start Hyperparameter Tuning Job .....	1765
Resource Limits for Automatic Model Tuning .....	1769
Best Practices for Hyperparameter Tuning .....	1770

Distributed Training .....	1771
Get Started with Distributed Training .....	1772
Basic Distributed Training Concepts .....	1772
Advanced Concepts .....	1773
Strategies .....	1774
Optimize Distributed Training .....	1775
Scenarios .....	1776
SageMaker Built-In Distributed Training Features .....	1779
Data Parallel Training .....	1779
Model Parallel Training .....	1795
Jupyter Notebook Examples .....	1819
Detect Posttraining Data and Model Bias .....	1821
Sample Notebooks .....	1821
Measure Posttraining Data and Model Bias .....	1821
Configure SageMaker Clarify processing jobs .....	1840
Run SageMaker Clarify Processing Jobs .....	1847
Troubleshoot Jobs .....	1850
Model Explainability .....	1852
Shapley Values .....	1853
SHAP Baselines for Explainability .....	1854
Create Feature Attribute Baselines and Explainability Reports .....	1854
Incremental Training .....	1855
Perform Incremental Training (Console) .....	1855
Perform Incremental Training (API) .....	1857
Managed Spot Training .....	1859
Using Managed Spot Training .....	1860
Managed Spot Training Lifecycle .....	1860
Use Checkpoints .....	1860
Frameworks and algorithms .....	1861
Enable Checkpointing .....	1862
Browse Checkpoint Files .....	1863
Resume Training From a Checkpoint .....	1863
Considerations for Checkpointing .....	1864
Use Augmented Manifest Files .....	1864
Augmented Manifest File format .....	1864
Stream Augmented Manifest File Data .....	1865
Use an Augmented Manifest File (Console) .....	1866
Use an Augmented Manifest File (API) .....	1867
Monitor and Analyze Using Metrics .....	1868
Sample Notebooks .....	1868
Defining Training Metrics .....	1868
Monitoring Training Job Metrics ( Console) .....	1871
Monitoring Training Job Metrics (SageMaker Console) .....	1871
Example: Viewing a Training and Validation Curve .....	1873
Inference .....	1875
Prerequisites .....	1875
What do you want to do? .....	1875
Manage Model Deployments .....	1876
Deploy Your Own Inference Code .....	1876
Guide to SageMaker .....	1876
Elastic Inference .....	1876
How EI Works .....	1877
Choose an EI Accelerator Type .....	1877
Use EI in a SageMaker Notebook Instance .....	1878
Use EI on a Hosted Endpoint .....	1878
Frameworks that Support EI .....	1878
Use EI with SageMaker Built-in Algorithms .....	1879

EI Sample Notebooks .....	1879
Set Up to Use EI .....	1879
Attach EI to a Notebook Instance .....	1882
Endpoints with Elastic Inference .....	1884
Batch Transform .....	1888
Use Batch Transform to Get Inferences from Large Datasets .....	1888
Speed up a Batch Transform Job .....	1889
Use Batch Transform to Test Production Variants .....	1889
Batch Transform Errors .....	1890
Sample Notebooks .....	1890
Associate Prediction Results with Input .....	1890
Multi-Model Endpoints .....	1895
Supported Algorithms and Frameworks .....	1896
Sample Notebooks .....	1896
How Amazon SageMaker multi-model endpoints work .....	1896
Setting SageMaker Multi-Model Endpoint Model Caching Behavior .....	1897
Instance Recommendations for Multi-Model Endpoint Deployments .....	1897
Create a Multi-Model Endpoint .....	1898
Invoke a Multi-Model Endpoint .....	1902
Add or Remove Models .....	1903
Bring Your Own Container .....	1904
Security .....	1909
CloudWatch Metrics for Multi-Model Endpoint Deployments .....	1909
Deploy multi-container endpoints .....	1910
Create a multi-container endpoint (Boto 3) .....	1910
Update a multi-container endpoint .....	1911
Delete a multi-container endpoint .....	1911
Use a multi-container endpoint with direct invocation .....	1911
Inference Pipelines .....	1917
Automatically Scale Models .....	1931
Prerequisites .....	1931
Configure model autoscaling with the console .....	1934
Register a model .....	1935
Define a scaling policy .....	1936
Apply a scaling policy .....	1938
Edit a scaling policy .....	1939
Delete a scaling policy .....	1940
Query Endpoint Autoscaling History .....	1942
Update or delete endpoints that use automatic scaling .....	1943
Load testing .....	1945
Use Amazon CloudFormation to update autoscaling policies .....	1946
Host Instance Storage Volumes .....	1946
Test models in production .....	1947
Test models by specifying traffic distribution .....	1947
Test models by invoking specific variants .....	1948
Model A/B test example .....	1949
Troubleshoot Deployments .....	1955
CPU Detection Errors with a JVM .....	1955
Deployment Best Practices .....	1956
Deploy Multiple Instances .....	1956
Model Monitor .....	1956
How It Works .....	1957
Monitor Data Quality .....	1958
Monitor Model Quality .....	1963
Monitor Bias Drift .....	1970
Monitor Feature Attribution Drift .....	1973
Capture Data .....	1977

Schedule Monitoring Jobs .....	1979
Prebuilt Container .....	1981
Interpret Results .....	1982
Visualize Results .....	1984
Advanced Topics .....	1990
Register and Deploy Models with Model Registry .....	2003
Create a Model Group .....	2003
Register a Model Version .....	2007
View Model Groups and Versions .....	2008
View Model Version Details .....	2010
Update Model Approval Status .....	2012
Deploy a Model in the Registry .....	2016
Deploy a Model Version from a Different Account .....	2017
View the Deployment History of a Model .....	2018
Compile and Deploy Models with Neo .....	2021
What is SageMaker Neo? .....	2021
How it Works .....	2021
Sample Notebooks .....	2022
Compile Models .....	2023
Cloud Instances .....	2036
Edge Devices .....	2063
Troubleshoot Errors .....	2080
SageMaker Edge Manager .....	2086
Why Use Edge Manager? .....	2086
How Does it Work? .....	2086
How Do I Use SageMaker Edge Manager? .....	2087
Getting Started .....	2087
Set Up Devices and Fleets .....	2100
Package Model .....	2107
Edge Manager Agent .....	2113
Manage Model .....	2125
Docker containers with SageMaker .....	2134
Scenarios and Guidance .....	2134
Docker Container Basics .....	2135
Use Prebuilt SageMaker Docker images .....	2135
Prebuilt Deep Learning Images .....	2135
Prebuilt Scikit-learn and Spark ML Images .....	2136
Deep Graph Networks .....	2138
Extend a Prebuilt Container .....	2140
Adapting Your Own Docker Container to Work with SageMaker .....	2148
Individual Framework Libraries .....	2149
SageMaker Training and Inference Toolkits .....	2149
Adapting Your Own Training Container .....	2151
Adapting Your Own Inference Container .....	2156
Create a container with your own algorithms and models .....	2160
Use Your Own Training Algorithms .....	2160
Use Your Own Inference Code .....	2166
Example Notebooks .....	2175
Setup .....	2175
Host Models Trained in Scikit-learn .....	2175
Package TensorFlow and Scikit-learn Models for Use in SageMaker .....	2175
Train and Deploy a Neural Network on SageMaker .....	2176
Training Using Pipe Mode .....	2176
Bring Your Own R Model .....	2176
Extend a Prebuilt PyTorch Container Image .....	2176
Train and Debug Training Jobs on a Custom Container .....	2176
Troubleshooting .....	2177

Workflows .....	2178
Amazon SageMaker Model Building Pipelines .....	2178
Pipeline Overview .....	2179
Create and Manage Pipelines .....	2201
Automate MLOps with SageMaker Projects .....	2232
What is a SageMaker Project? .....	2233
Why Should You Use MLOps? .....	2234
SageMaker Studio Permissions Required to Use Projects .....	2236
Create an MLOps Project .....	2236
MLOps Project Templates .....	2240
View Project Resources .....	2242
SageMaker MLOps Project Walkthrough .....	2244
ML Lineage Tracking .....	2256
Tracking Entities .....	2257
SageMaker Created Entities .....	2258
Manually Create Entities .....	2260
Kubernetes Orchestration .....	2263
SageMaker Operators for Kubernetes .....	2263
SageMaker Components for Kubeflow Pipelines .....	2294
Augmented AI .....	2309
Get Started with Amazon Augmented AI .....	2310
Core Components of Amazon A2I .....	2310
Prerequisites to Using Augmented AI .....	2314
Tutorial: Get Started in the Amazon A2I Console .....	2314
Tutorial: Get Started Using the Amazon A2I API .....	2320
Use Cases and Examples .....	2330
Use SageMaker Notebook Instance with Amazon A2I Jupyter Notebook .....	2332
Use with Amazon Textract .....	2332
Use with Amazon Rekognition .....	2335
Use With Custom Task Types .....	2337
Create a Human Review Workflow .....	2339
Create a Human Review Workflow (Console) .....	2340
Create a Human Review Workflow (API) .....	2341
JSON Schema for Human Loop Activation Conditions in Amazon Augmented AI .....	2345
Delete a Human Review Workflow .....	2357
Delete a Flow Definition Using the Console or the SageMaker API .....	2358
Create and Start a Human Loop .....	2358
Create and Start a Human Loop for a Built-in Task Type .....	2359
Create and Start a Human Loop for a Custom Task Type .....	2362
Next Steps: .....	2363
Delete a Human Loop .....	2363
Human Loop Data Retention and Deletion .....	2364
Stop and Delete a Flow Definition Using the Console or the Amazon A2I API .....	2364
Create and Manage Worker Task Templates .....	2366
Create and Delete Worker Task Templates .....	2366
Create Custom Worker Task Templates .....	2368
Creating Good Worker Instructions .....	2375
Monitor and Manage Your Human Loop .....	2376
Output Data .....	2377
Output Data From Built-In Task Types .....	2377
Output Data From Custom Task Types .....	2384
Track Worker Activity .....	2385
Permissions and Security .....	2386
CORS Permission Requirement .....	2387
Add Permissions to the IAM Role Used to Create a Flow Definition .....	2388
Create an IAM User That Can Invoke Amazon A2I API Operations .....	2389

Create an IAM User With Permissions to Invoke Amazon A2I, Amazon Textract, and Amazon Rekognition API Operations .....	2390
Enable Worker Task Template Previews .....	2390
Using Amazon A2I with Amazon KMS Encrypted Buckets .....	2391
Additional Permissions and Security Resources .....	2392
CloudWatch Events .....	2392
Send Events from Your Human Loop to CloudWatch Events .....	2393
Set Up a Target to Process Events .....	2393
Use Human Review Output .....	2394
More Information .....	2394
API References .....	2394
Programmatic Tutorials .....	2394
Marketplace .....	2396
Use your own algorithms and models with the Amazon Marketplace .....	2396
Create Algorithm and Model Package Resources .....	2396
Use Algorithm and Model Package Resources .....	2402
Security .....	2410
Access Control .....	2410
Access control and Studio notebooks .....	2410
Control root access to a Notebook instance .....	2412
Data Protection .....	2412
Protect Data at Rest Using Encryption .....	2413
Protecting Data in Transit with Encryption .....	2414
Key Management .....	2416
Internetwork Traffic Privacy .....	2417
Identity and Access Management .....	2417
Audience .....	2417
Authenticating with Identities .....	2418
Managing Access Using Policies .....	2420
How Amazon SageMaker Works with IAM .....	2421
Identity-Based Policy Examples .....	2424
SageMaker Roles .....	2447
Amazon Managed Policies for SageMaker .....	2464
Amazon SageMaker API Permissions Reference .....	2474
Troubleshooting .....	2487
Logging and Monitoring .....	2489
Compliance Validation .....	2490
Resilience .....	2490
Infrastructure Security .....	2491
Connect Studio Notebooks to Resources in a VPC .....	2491
Connect a Notebook Instance to Resources in a VPC .....	2493
Training and Inference Containers Run in Internet-Free Mode .....	2494
SageMaker Scans Amazon Web Services Marketplace Training and Inference Containers for Security Vulnerabilities .....	2495
Connect to SageMaker Through a VPC Interface Endpoint .....	2495
Give SageMaker Compilation Jobs Access to Resources in Your Amazon VPC .....	2504
Give SageMaker Processing Jobs Access to Resources in Your Amazon VPC .....	2507
Give SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC .....	2510
Give SageMaker Training Jobs Access to Resources in Your Amazon VPC .....	2513
Give Batch Transform Jobs Access to Resources in Your Amazon VPC .....	2516
Give Amazon SageMaker Clarify Jobs Access to Resources in Your Amazon VPC .....	2519
Monitoring .....	2523
Monitoring with CloudWatch .....	2523
Endpoint Invocation Metrics .....	2524
Multi-Model Endpoint Metrics .....	2525
Jobs and Endpoint Metrics .....	2527
Ground Truth Metrics .....	2529

Feature Store Metrics .....	2531
Pipelines Metrics .....	2532
Logging with CloudWatch .....	2534
Log SageMaker API Calls with CloudTrail .....	2535
SageMaker Information in CloudTrail .....	2535
Operations Performed by Automatic Model Tuning .....	2536
Understanding SageMaker Log File Entries .....	2536
Automating with EventBridge .....	2537
Training job state change .....	2538
HyperParameter tuning job state change .....	2539
Transform job state change .....	2541
Endpoint state change .....	2541
Feature group state change .....	2542
Model package state change .....	2543
Pipeline execution state change .....	2544
Pipeline step state change .....	2544
Availability Zones .....	2546
API and SDK Reference .....	2547
Overview .....	2547
Programming Model for Amazon SageMaker .....	2547
Document History .....	2549
Amazon glossary .....	2551

# What Is Amazon SageMaker?

Amazon SageMaker is a fully managed machine learning service. With SageMaker, data scientists and developers can quickly and easily build and train machine learning models, and then directly deploy them into a production-ready hosted environment. It provides an integrated Jupyter authoring notebook instance for easy access to your data sources for exploration and analysis, so you don't have to manage servers. It also provides common machine learning algorithms that are optimized to run efficiently against extremely large data in a distributed environment. With native support for bring-your-own-algorithms and frameworks, SageMaker offers flexible distributed training options that adjust to your specific workflows. Deploy a model into a secure and scalable environment by launching it with a few clicks from SageMaker Studio or the SageMaker console. Training and hosting are billed by minutes of usage, with no minimum fees and no upfront commitments.

This guide includes information and tutorials on SageMaker features. For additional information, see [Amazon SageMaker developer resources](#).

## Topics

- [Amazon SageMaker Features \(p. 1\)](#)
- [Amazon SageMaker Pricing \(p. 3\)](#)
- [Are You a First-time User of Amazon SageMaker? \(p. 3\)](#)

## Amazon SageMaker Features

Amazon SageMaker includes the following features:

### [SageMaker Studio \(p. 69\)](#)

An integrated machine learning environment where you can build, train, deploy, and analyze your models all in the same application.

### [SageMaker Model Registry \(p. 2003\)](#)

Versioning, artifact and lineage tracking, approval workflow, and cross account support for deployment of your machine learning models.

### [SageMaker Projects \(p. 2232\)](#)

Create end-to-end ML solutions with CI/CD by using SageMaker projects.

### [SageMaker Model Building Pipelines \(p. 2178\)](#)

Create and manage machine learning pipelines integrated directly with SageMaker jobs.

### [SageMaker ML Lineage Tracking \(p. 2256\)](#)

Track the lineage of machine learning workflows.

### [SageMaker Data Wrangler \(p. 577\)](#)

Import, analyze, prepare, and featurize data in SageMaker Studio. You can integrate Data Wrangler into your machine learning workflows to simplify and streamline data pre-processing and feature engineering using little to no coding. You can also add your own Python scripts and transformations to customize your data prep workflow.

### [SageMaker Feature Store \(p. 673\)](#)

A centralized store for features and associated metadata so features can be easily discovered and reused. You can create two types of stores, an Online or Offline store. The Online Store can be used

for low latency, real-time inference use cases and the Offline Store can be used for training and batch inference.

#### [SageMaker JumpStart \(p. 40\)](#)

Learn about SageMaker features and capabilities through curated 1-click solutions, example notebooks, and pretrained models that you can deploy. You can also fine-tune the models and deploy them.

#### [SageMaker Clarify \(p. 6\)](#)

Improve your machine learning models by detecting potential bias and help explain the predictions that models make.

#### [SageMaker Edge Manager \(p. 2086\)](#)

Optimize custom models for edge devices, create and manage fleets and run models with an efficient runtime.

#### [SageMaker Ground Truth \(p. 165\)](#)

High-quality training datasets by using workers along with machine learning to create labeled datasets.

#### [Amazon Augmented AI \(p. 2309\)](#)

Build the workflows required for human review of ML predictions. Amazon A2I brings human review to all developers, removing the undifferentiated heavy lifting associated with building human review systems or managing large numbers of human reviewers.

#### [SageMaker Studio Notebooks \(p. 76\)](#)

The next generation of SageMaker notebooks that include Amazon Web Services Single Sign On (Amazon Web Services SSO) integration, fast start-up times, and single-click sharing.

The next generation of SageMaker notebooks that include fast start-up times and single-click sharing.

#### [SageMaker Experiments \(p. 1553\)](#)

Experiment management and tracking. You can use the tracked data to reconstruct an experiment, incrementally build on experiments conducted by peers, and trace model lineage for compliance and audit verifications.

#### [SageMaker Debugger \(p. 1579\)](#)

Inspect training parameters and data throughout the training process. Automatically detect and alert users to commonly occurring errors such as parameter values getting too large or small.

#### [SageMaker Autopilot \(p. 142\)](#)

Users without machine learning knowledge can quickly build classification and regression models.

#### [SageMaker Model Monitor \(p. 1956\)](#)

Monitor and analyze models in production (endpoints) to detect data drift and deviations in model quality.

#### [SageMaker Neo \(p. 2021\)](#)

Train machine learning models once, then run anywhere in the cloud and at the edge.

#### [SageMaker Elastic Inference \(p. 1876\)](#)

Speed up the throughput and decrease the latency of getting real-time inferences.

#### [Reinforcement Learning \(p. 1552\)](#)

Maximize the long-term reward that an agent receives as a result of its actions.

### **Preprocessing (p. 664)**

Analyze and preprocess data, tackle feature engineering, and evaluate models.

### **Batch Transform (p. 1888)**

Preprocess datasets, run inference when you don't need a persistent endpoint, and associate input records with inferences to assist the interpretation of results.

## Amazon SageMaker Pricing

As with other Amazon products, there are no contracts or minimum commitments for using Amazon SageMaker. For more information about the cost of using SageMaker, see [SageMaker Pricing](#).

## Are You a First-time User of Amazon SageMaker?

If you are a first-time user of SageMaker, we recommend that you do the following:

1. **Read How Amazon SageMaker Works (p. 3)** – This section provides an overview of SageMaker, explains key concepts, and describes the core components involved in building AI solutions with SageMaker. We recommend that you read this topic in the order presented.
2. **Set Up Amazon SageMaker (p. 34)** – This section explains how to set up your Amazon account and onboard to SageMaker Studio.
3. Amazon SageMaker Autopilot simplifies the machine learning experience by automating machine learning tasks. If you are new to SageMaker, it provides the easiest learning path. It also serves as an excellent ML learning tool that provides visibility into the code with notebooks generated for each of the automated ML tasks. For an introduction to its capabilities, see [Automate model development with Amazon SageMaker Autopilot \(p. 142\)](#). To get started building, training, and deploying machine learning models, Autopilot provides:
  - [Samples: Explore modeling with Amazon SageMaker Autopilot \(p. 143\)](#)
  - [Videos: Use Autopilot to automate and explore the machine learning process \(p. 144\)](#)
  - [Tutorials: Get started with Amazon SageMaker Autopilot \(p. 145\)](#)
4. **Get Started with Amazon SageMaker (p. 34)** – This section walks you through training your first model using SageMaker Studio, or the SageMaker console and the SageMaker API. You use training algorithms provided by SageMaker.
5. **Explore other topics** – Depending on your needs, do the following:
  - **Submit Python code to train with deep learning frameworks** – In SageMaker, you can use your own training scripts to train models. For information, see [Use Machine Learning Frameworks, Python, and R with Amazon SageMaker \(p. 16\)](#).
  - **Use SageMaker directly from Apache Spark** – For information, see [Use Apache Spark with Amazon SageMaker \(p. 17\)](#).
  - **Use SageMaker to train and deploy your own custom algorithms** – Package your custom algorithms with Docker so you can train and/or deploy them in SageMaker. To learn how SageMaker interacts with Docker containers, and for the SageMaker requirements for Docker images, see [Using Docker containers with SageMaker \(p. 2134\)](#).
6. **View the API Reference** – This section describes the SageMaker API operations.

## How Amazon SageMaker Works

SageMaker is a fully managed service that enables you to quickly and easily integrate machine learning-based models into your applications. This section provides an overview of machine learning and explains

how SageMaker works. If you are a first-time user of SageMaker, we recommend that you read the following sections in order:

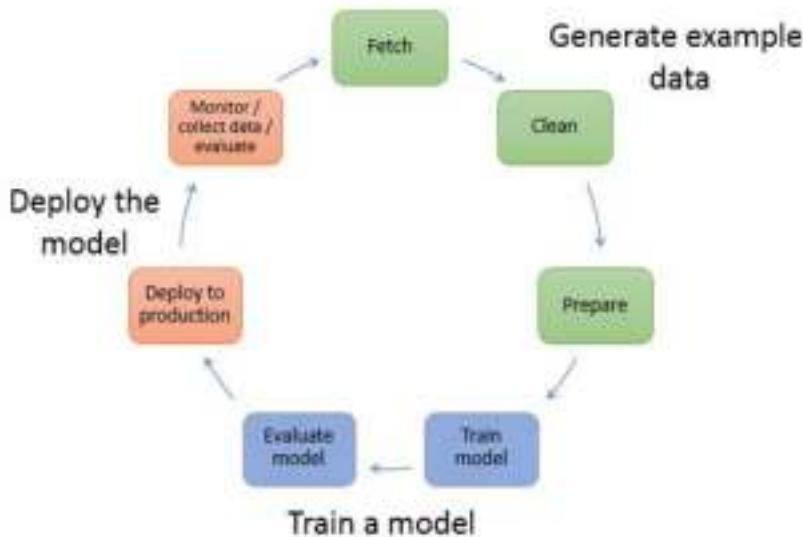
1. [Machine Learning with Amazon SageMaker \(p. 4\)](#)
2. [Explore, Analyze, and Process Data \(p. 5\)](#)
3. [Train a Model with Amazon SageMaker \(p. 8\)](#)
4. [Deploy a Model in Amazon SageMaker \(p. 10\)](#)
5. [Use Machine Learning Frameworks, Python, and R with Amazon SageMaker \(p. 16\)](#)
6. [Get Started with Amazon SageMaker \(p. 34\)](#)

## Machine Learning with Amazon SageMaker

This section describes a typical machine learning workflow and summarizes how you accomplish those tasks with Amazon SageMaker.

In machine learning, you "teach" a computer to make predictions, or inferences. First, you use an algorithm and example data to train a model. Then you integrate your model into your application to generate inferences in real time and at scale. In a production environment, a model typically learns from millions of example data items and produces inferences in hundreds to less than 20 milliseconds.

The following diagram illustrates the typical workflow for creating a machine learning model:



As the diagram illustrates, you typically perform the following activities:

1. **Generate example data**—To train a model, you need example data. The type of data that you need depends on the business problem that you want the model to solve (the inferences that you want the model to generate). For example, suppose that you want to create a model to predict a number given an input image of a handwritten digit. To train such a model, you need example images of handwritten numbers.

Data scientists often spend a lot of time exploring and preprocessing, or "wrangling," example data before using it for model training. To preprocess data, you typically do the following:

- a. **Fetch the data**— You might have in-house example data repositories, or you might use datasets that are publicly available. Typically, you pull the dataset or datasets into a single repository.

- b. **Clean the data**—To improve model training, inspect the data and clean it as needed. For example, if your data has a `country_name` attribute with values `United States` and `US`, you might want to edit the data to be consistent.
- c. **Prepare or transform the data**—To improve performance, you might perform additional data transformations. For example, you might choose to combine attributes. If your model predicts the conditions that require de-icing an aircraft, instead of using temperature and humidity attributes separately, you might combine those attributes into a new attribute to get a better model.

In SageMaker, you preprocess example data in a Jupyter notebook on your notebook instance. You use your notebook to fetch your dataset, explore it, and prepare it for model training. For more information, see [Explore, Analyze, and Process Data \(p. 5\)](#). For more information about preparing data in Amazon Marketplace, see [data preparation](#).

**2. Train a model**—Model training includes both training and evaluating the model, as follows:

- **Training the model**— To train a model, you need an algorithm. The algorithm you choose depends on a number of factors. For a quick, out-of-the-box solution, you might be able to use one of the algorithms that SageMaker provides. For a list of algorithms provided by SageMaker and related considerations, see [Use Amazon SageMaker Built-in Algorithms \(p. 718\)](#).

You also need compute resources for training. Depending on the size of your training dataset and how quickly you need the results, you can use resources ranging from a single general-purpose instance to a distributed cluster of GPU instances. For more information, see [Train a Model with Amazon SageMaker \(p. 8\)](#).

- **Evaluating the model**—After you've trained your model, you evaluate it to determine whether the accuracy of the inferences is acceptable. In SageMaker, you use either the Amazon SDK for Python (Boto) or the high-level Python library that SageMaker provides to send requests to the model for inferences.

You use a Jupyter notebook in your SageMaker notebook instance to train and evaluate your model.

**3. Deploy the model**— You traditionally re-engineer a model before you integrate it with your application and deploy it. With SageMaker hosting services, you can deploy your model independently, decoupling it from your application code. For more information, see [Deploy a Model on SageMaker Hosting Services \(p. 10\)](#).

Machine learning is a continuous cycle. After deploying a model, you monitor the inferences, collect "ground truth," and evaluate the model to identify drift. You then increase the accuracy of your inferences by updating your training data to include the newly collected ground truth. You do this by retraining the model with the new dataset. As more and more example data becomes available, you continue retraining your model to increase accuracy.

## Explore, Analyze, and Process Data

Before using a dataset to train a model, data scientists typically explore, analyze, and preprocess it.

To pre-process data, use one of the following methods:

- Use a Jupyter notebook on an Amazon SageMaker notebook instance to do the following:
  - Write code to create model training jobs
  - Deploy models to SageMaker hosting
  - Test or validate your models

For a Get Started guide about training and deploying a model in the SageMaker Notebook instances, see [Get Started with Amazon SageMaker Notebook Instances \(p. 51\)](#).

For more information about the SageMaker Notebook instances, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#).

- You can use a model to transform data by using SageMaker batch transform. For more information, see [\(Optional\) Make Prediction with Batch Transform \(p. 64\)](#).

Amazon SageMaker Processing enables running jobs to preprocess and postprocess data, perform feature engineering, and evaluate models on SageMaker easily and at scale. When combined with the other critical machine learning tasks provided by SageMaker, such as training and hosting, Processing provides you with the benefits of a fully managed machine learning environment, including all the security and compliance support built into SageMaker. With Processing, you have the flexibility to use the built-in data processing containers or to bring your own containers and submit custom jobs to run on managed infrastructure. After you submit a job, SageMaker launches the compute instances, processes and analyzes the input data, and releases the resources upon completion. For more information, see [Process Data \(p. 664\)](#).

- For information about how to run your own data processing scripts, see [Data Processing with scikit-learn \(p. 666\)](#).
- For information about how to build your own processing container to run scripts, see [Build Your Own Processing Container \(Advanced Scenario\) \(p. 668\)](#).

## What Is Fairness and Model Explainability for Machine Learning Predictions?

Amazon SageMaker Clarify helps improve your machine learning (ML) models by detecting potential bias and helping explain the predictions that models make. It helps you identify various types of bias in pretraining data and in posttraining that can emerge during model training or when the model is in production. SageMaker Clarify helps explain how these models make predictions using a feature attribution approach. It also monitors inferences models make in production for bias or feature attribution drift. The fairness and explainability functionality provided by SageMaker Clarify provides components that help Amazon customers build less biased and more understandable machine learning models. It also provides tools to help you generate model governance reports that you can use to inform risk and compliance teams, and external regulators.

Machine learning models and data-driven systems are being increasingly used to help make decisions across domains such as financial services, healthcare, education, and human resources. Machine learning applications provide benefits such as improved accuracy, increased productivity, and cost savings to help meet regulatory requirements, improve business decisions, and provide better insights into data science procedures.

- **Regulatory** – In many situations, it is important to understand why an ML model made a specific prediction and also whether the prediction it made was impacted by any bias, either during training or at inference. Recently, policymakers, regulators, and advocates have raised awareness about the ethical and policy challenges posed by ML and data-driven systems. In particular, they have expressed concerns about the potentially discriminatory impact of such systems (for example, inadvertently encoding of bias into automated decisions).
- **Business** – The adoption of AI systems in regulated domains requires trust, which can be built by providing reliable explanations of the behavior of trained models and how the deployed models make predictions. Model explainability may be particularly important to certain industries with reliability, safety, and compliance requirements, such as financial services, human resources, healthcare, and

automated transportation. To take a common financial example, lending applications that incorporate the use of ML models might need to provide explanations about how those models made certain predictions to internal teams of loan officers, customer service representatives, and forecasters, in addition to end users/customers.

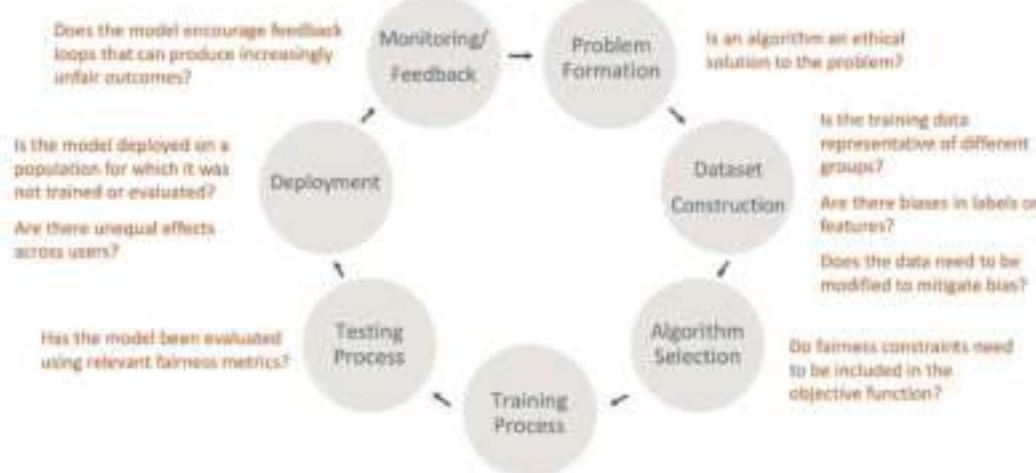
- **Data Science** – Data scientists and ML engineers need tools to generate the insights required to debug and improve ML models through better feature engineering, to determine whether a model is making inferences based on noisy or irrelevant features, and to understand the limitations of their models and failure modes their models may encounter.

For a blog that shows how to architect and build a complete machine learning use case involving fraudulent automobile claims that integrates SageMaker Clarify into a SageMaker pipeline, see the [Architect and build the full machine learning lifecycle with Amazon: An end-to-end Amazon SageMaker demo](#). This blog discusses how to assess pre and post training bias, how to mitigate the bias, and how the data features impact the prediction. There are links to the relevant code for each task in the ML lifecycle, including the creation of an automated workflow that integrates the fairness and explainability functionality of SageMaker Clarify into a SageMaker Pipeline.

## Best Practices for Evaluating Fairness and Explainability in the ML Lifecycle

**Fairness as a Process** – The notions of bias and fairness are highly dependent on the application and the choice of attributes for which bias is to be measured, in addition to the choice of bias metrics. These choices should be guided by ethical, business, and regulatory considerations. Building consensus and achieving collaboration across key stakeholders (such as product, policy, legal, public relations (PR), engineering, AI/ML teams, end users, and communities) is important for the successful adoption of fair and transparent ML applications.

**Fairness and Explainability by Design in the ML Lifecycle** – You should consider fairness and explainability during each stage of the ML lifecycle: problem formation, dataset construction, algorithm selection, model training process, testing process, deployment, and monitoring/feedback. It is important to have the right tools to do this analysis. To encourage engaging with these considerations, here are a few example questions we recommend you ask during each of these stages.



## Sample Notebooks

Amazon SageMaker Clarify provides the following sample notebooks:

- [Explainability and bias detection with Amazon SageMaker Clarify](#) – Use SageMaker Clarify to create a processing job for the detecting bias and explaining model predictions with feature attributions.
- [Monitoring bias drift and feature attribution drift Amazon SageMaker Clarify](#) – Use Amazon SageMaker Model Monitor to monitor bias drift and feature attribution drift over time.
- [Mitigate Bias, Train another unbiased Model and Put in the Model Registry](#) – This notebook describes how to detect bias using SageMaker Clarify, mitigate it with [Synthetic Minority Over-sampling Technique \(SMOTE\)](#), train another model, then put it in the Model Registry along with all the lineage of the artifacts created along the way: data, code and model metadata. This notebook forms part of a series that shows how to integrate SageMaker Clarify into a SageMaker Pipeline that is described in the [Architect and build the full machine learning lifecycle with Amazon](#) blog.

These notebooks have been verified to run in Amazon SageMaker Studio only. If you need instructions on how to open a notebook in Amazon SageMaker Studio, see [Create or Open an Amazon SageMaker Studio Notebook \(p. 79\)](#). If you're prompted to choose a kernel, choose **Python 3 (Data Science)**.

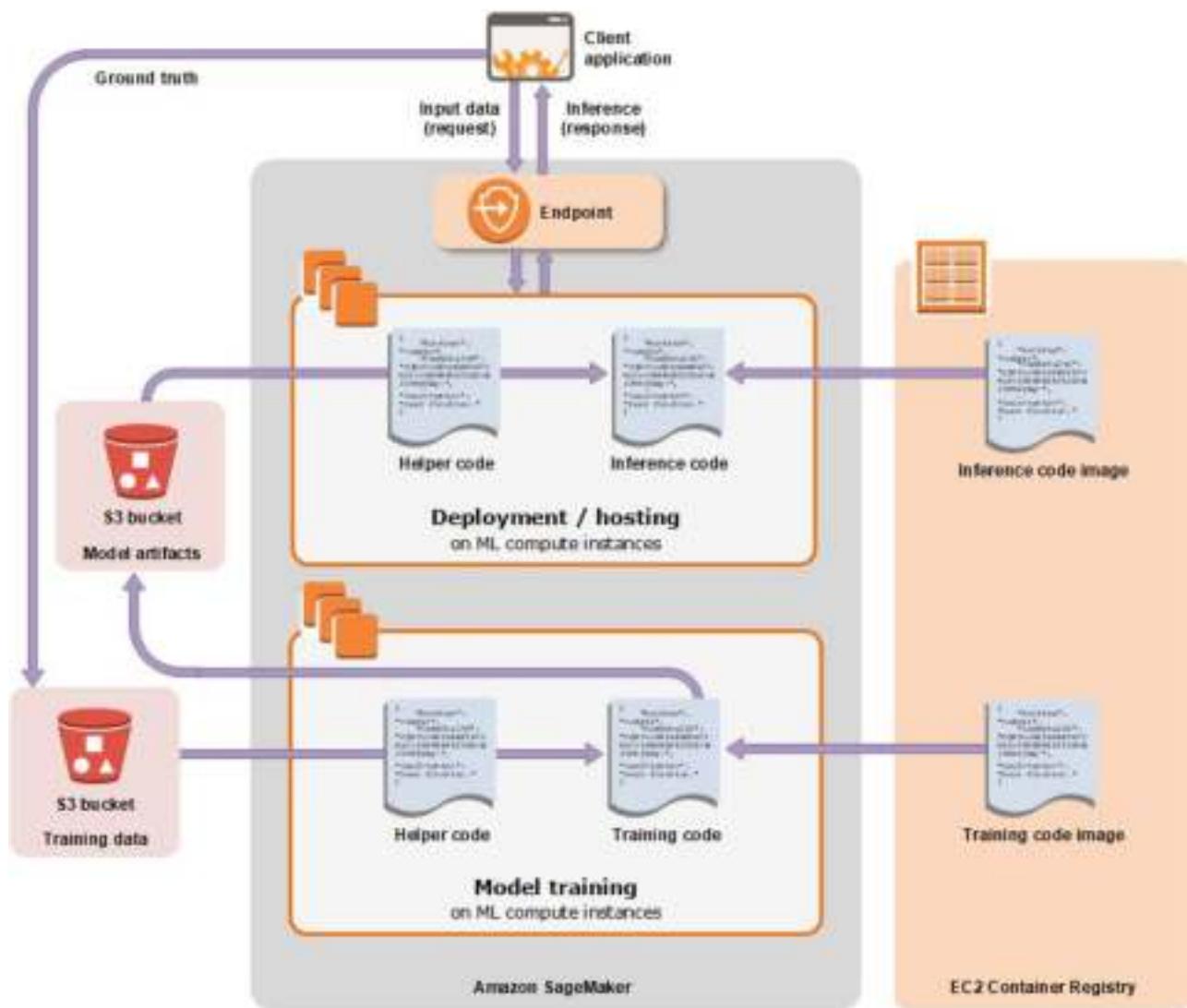
## Guide to the SageMaker Clarify Documentation

Bias can occur and be measured in the data at each stage of the machine learning lifecycle: before training a model and after model training. SageMaker Clarify can provide feature attribution explanations of model predictions for trained models and for models deployed to production, where models can be monitored for any drift from their baseline explanatory attributions. The documentation for SageMaker Clarify is embedding throughout the larger SageMaker documentation set at the relevant ML stages as follows:

- For further information on detecting bias in preprocessing data before it's used to train a model, see [Detect Pretraining Data Bias \(p. 561\)](#).
- For further information on detecting posttraining data and model bias, see [Detect Posttraining Data and Model Bias \(p. 1821\)](#).
- For further information on the model-agnostic feature attribution approach to explain model predictions after training, see [Model Explainability \(p. 1852\)](#).
- For further information on monitoring for bias in production model inferences due to the drift of data away from the baseline used to train the model, see [Monitor Bias Drift for Models in Production \(p. 1970\)](#).
- For further information on monitoring for the drift of features' contributions away from the baseline that was established during model training, see [Monitor Feature Attribution Drift for Models in Production \(p. 1973\)](#).

## Train a Model with Amazon SageMaker

The following diagram shows how you train and deploy a model with Amazon SageMaker:



The area labeled SageMaker highlights the two components of SageMaker: model training and model deployment.

To train a model in SageMaker, you create a training job. The training job includes the following information:

- The URL of the Amazon Simple Storage Service (Amazon S3) bucket where you've stored the training data.
- The compute resources that you want SageMaker to use for model training. Compute resources are ML compute instances that are managed by SageMaker.
- The URL of the S3 bucket where you want to store the output of the job.
- The Amazon Elastic Container Registry path where the training code is stored. For more information, see [Docker Registry Paths and Example Code \(p. 725\)](#).

You have the following options for a training algorithm:

- **Use an algorithm provided by SageMaker**—SageMaker provides training algorithms. If one of these meets your needs, it's a great out-of-the-box solution for quick model training. For a list of algorithms

provided by SageMaker, see [Use Amazon SageMaker Built-in Algorithms \(p. 718\)](#). To try an exercise that uses an algorithm provided by SageMaker, see [Get Started with Amazon SageMaker \(p. 34\)](#).

- **Use SageMaker Debugger**—to inspect training parameters and data throughout the training process when working with the TensorFlow, PyTorch, and Apache MXNet learning frameworks or the XGBoost algorithm. Debugger automatically detects and alerts users to commonly occurring errors such as parameter values getting too large or small. For more information about using Debugger, see [Amazon SageMaker Debugger \(p. 1579\)](#). Debugger sample notebooks are available at [Amazon SageMaker Debugger Samples](#).
- **Use Apache Spark with SageMaker**—SageMaker provides a library that you can use in Apache Spark to train models with SageMaker. Using the library provided by SageMaker is similar to using Apache Spark MLlib. For more information, see [Use Apache Spark with Amazon SageMaker \(p. 17\)](#).
- **Submit custom code to train with deep learning frameworks**—You can submit custom Python code that uses TensorFlow, PyTorch, or Apache MXNet for model training. For more information, see [Use TensorFlow with Amazon SageMaker \(p. 31\)](#), [Use PyTorch with Amazon SageMaker \(p. 27\)](#), and [Use Apache MXNet with Amazon SageMaker \(p. 16\)](#).
- **Use your own custom algorithms**—Put your code together as a Docker image and specify the registry path of the image in a SageMaker CreateTrainingJob API call. For more information, see [Using Docker containers with SageMaker \(p. 2134\)](#).
- 

After you create the training job, SageMaker launches the ML compute instances and uses the training code and the training dataset to train the model. It saves the resulting model artifacts and other output in the S3 bucket you specified for that purpose.

You can create a training job with the SageMaker console or the API. For information about creating a training job with the API, see the [CreateTrainingJob](#) API.

When you create a training job with the API, SageMaker replicates the entire dataset on ML compute instances by default. To make SageMaker replicate a subset of the data on each ML compute instance, you must set the `S3DataDistributionType` field to `ShardedByS3Key`. You can set this field using the low-level SDK. For more information, see `S3DataDistributionType` in [S3DataSource](#).

**Important**

To prevent your algorithm container from contending for memory, we reserve memory for our SageMaker critical system processes on your ML compute instances and therefore you cannot expect to see all the memory for your instance type.

## Deploy a Model in Amazon SageMaker

After you train your model, you can deploy it using Amazon SageMaker to get predictions in any of the following ways:

- To set up a persistent endpoint to get one prediction at a time, use SageMaker hosting services.
- To get predictions for an entire dataset, use SageMaker batch transform.

**Topics**

- [Deploy a Model on SageMaker Hosting Services \(p. 10\)](#)

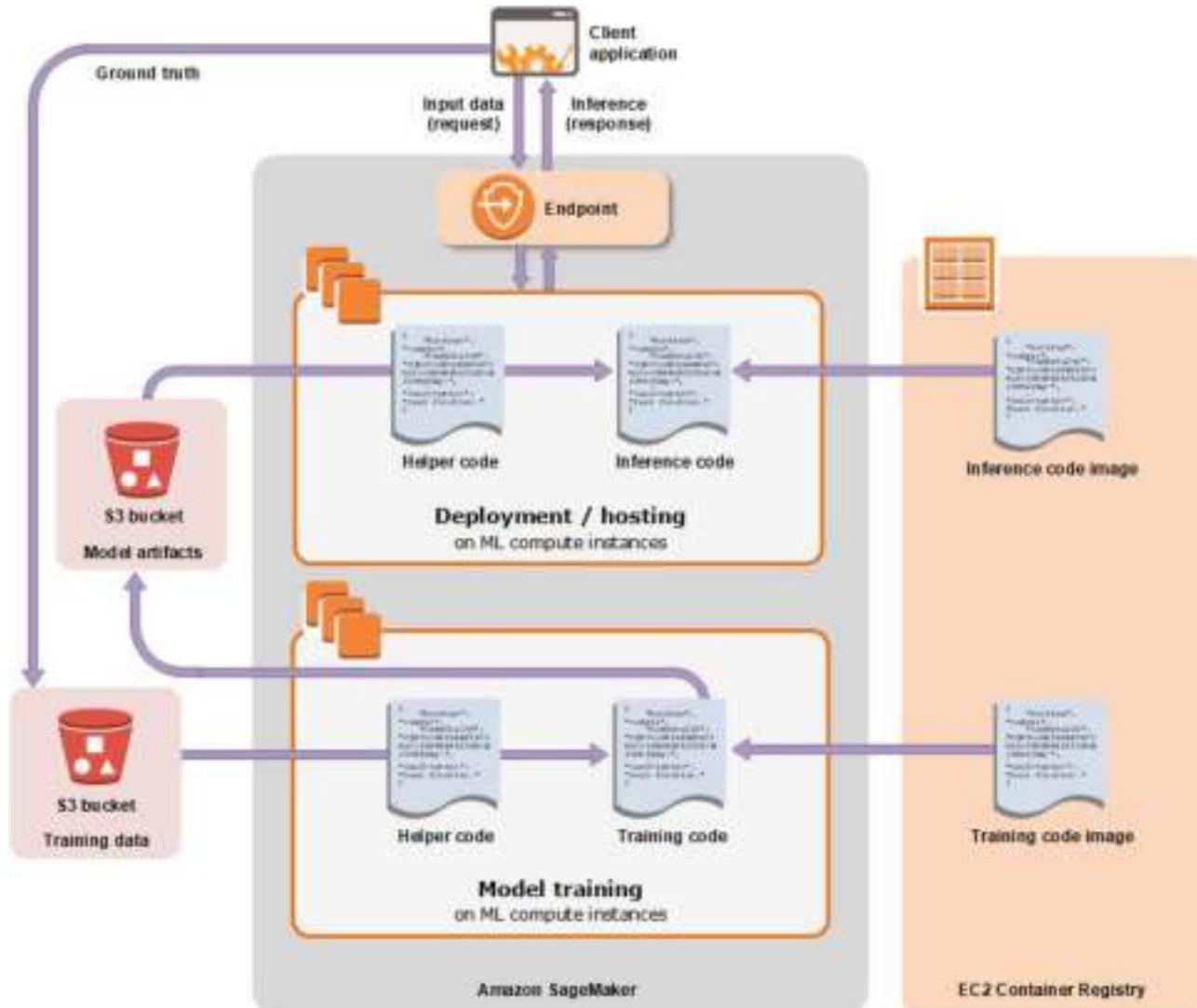
## Deploy a Model on SageMaker Hosting Services

For an example of how to deploy a model to the SageMaker hosting service, see [Deploy the Model to SageMaker Hosting Services \(p. 63\)](#).

Or, if you prefer, watch the following video tutorial:

[Deploy Your ML Models to Production at Scale with Amazon SageMaker](#)

SageMaker provides model hosting services for model deployment, as shown in the following diagram. SageMaker provides an HTTPS endpoint where your machine learning model is available to provide inferences.



Deploying a model using SageMaker hosting services is a three-step process:

1. **Create a model in SageMaker**—By creating a model, you tell SageMaker where it can find the model components. This includes the S3 path where the model artifacts are stored and the Docker registry path for the image that contains the inference code. In subsequent deployment steps, you specify the model by name. For more information, see the [CreateModel API](#).

**Note**

The S3 bucket where the model artifacts are stored must be in the same region as the model that you are creating.

2. **Create an endpoint configuration for an HTTPS endpoint**—You specify the name of one or more models in production variants and the ML compute instances that you want SageMaker to launch to host each production variant.

**Note**

SageMaker supports running (a) multiple models, (b) multiple variants of a model, or (c) combinations of models and variants on the same endpoint. Model variants can reference the same model inference container (i.e. run the same algorithm), but use different model artifacts (e.g., different model weight values based on other hyper-parameter configurations). In contrast, two different models may use the same algorithm, but focus on different business problems or underlying goals and may operate on different data sets.

When hosting models in production, you can configure the endpoint to elastically scale the deployed ML compute instances. For each production variant, you specify the number of ML compute instances that you want to deploy. When you specify two or more instances, SageMaker launches them in multiple Availability Zones. This ensures continuous availability. SageMaker manages deploying the instances. For more information, see the [CreateEndpointConfig API](#).

3. **Create an HTTPS endpoint**—Provide the endpoint configuration to SageMaker. The service launches the ML compute instances and deploys the model or models as specified in the configuration. For more information, see the [CreateEndpoint API](#). To get inferences from the model, client applications send requests to the SageMaker Runtime HTTPS endpoint. For more information about the API, see the [InvokeEndpoint API](#).

**Note**

Endpoints are scoped to an individual Amazon account, and are not public. The URL does not contain the account ID, but SageMaker determines the account ID from the authentication token that is supplied by the caller.

For an example of how to use Amazon API Gateway and Amazon Lambda to set up and deploy a web service that you can call from a client application that is not within the scope of your account, see [Call a SageMaker model endpoint using Amazon API Gateway and Amazon Lambda](#) in the *Amazon Machine Learning Blog*.

**Note**

When you create an endpoint, SageMaker attaches an Amazon EBS storage volume to each ML compute instance that hosts the endpoint. The size of the storage volume depends on the instance type. For a list of instance types that SageMaker hosting service supports, see [Amazon Service Limits](#). For a list of the sizes of the storage volumes that SageMaker attaches to each instance, see [Host Instance Storage Volumes \(p. 1946\)](#).

To increase a model's accuracy, you might choose to save the user's input data and ground truth, if available, as part of the training data. You can then retrain the model periodically with a larger, improved training dataset.

## Best Practices for Deploying Models on SageMaker Hosting Services

When hosting models using SageMaker hosting services, consider the following:

- Typically, a client application sends requests to the SageMaker HTTPS endpoint to obtain inferences from a deployed model. You can also send requests to this endpoint from your Jupyter notebook during testing.
- You can deploy a model trained with SageMaker to your own deployment target. To do that, you need to know the algorithm-specific format of the model artifacts that were generated by model training. For more information about output formats, see the section corresponding to the algorithm you are using in [Common Data Formats for Training \(p. 1354\)](#).

- You can deploy multiple variants of a model to the same SageMaker HTTPS endpoint. This is useful for testing variations of a model in production. For example, suppose that you've deployed a model into production. You want to test a variation of the model by directing a small amount of traffic, say 5%, to the new model. To do this, create an endpoint configuration that describes both variants of the model. You specify the `ProductionVariant` in your request to the `CreateEndpointConfig`. For more information, see [ProductionVariant](#).
- You can configure a `ProductionVariant` to use Application Auto Scaling. For information about configuring automatic scaling, see [Automatically Scale Amazon SageMaker Models \(p. 1931\)](#).
- You can modify an endpoint without taking models that are already deployed into production out of service. For example, you can add new model variants, update the ML Compute instance configurations of existing model variants, or change the distribution of traffic among model variants. To modify an endpoint, you provide a new endpoint configuration. SageMaker implements the changes without any downtime. For more information see, [UpdateEndpoint](#) and [UpdateEndpointWeightsAndCapacities](#).
- Changing or deleting model artifacts or changing inference code after deploying a model produces unpredictable results. If you need to change or delete model artifacts or change inference code, modify the endpoint by providing a new endpoint configuration. Once you provide the new endpoint configuration, you can change or delete the model artifacts corresponding to the old endpoint configuration.
- If you want to get inferences on entire datasets, consider using batch transform as an alternative to hosting services. For information, see [Get Inferences for an Entire Dataset with Batch Transform \(p. 13\)](#)

## Get Inferences for an Entire Dataset with Batch Transform

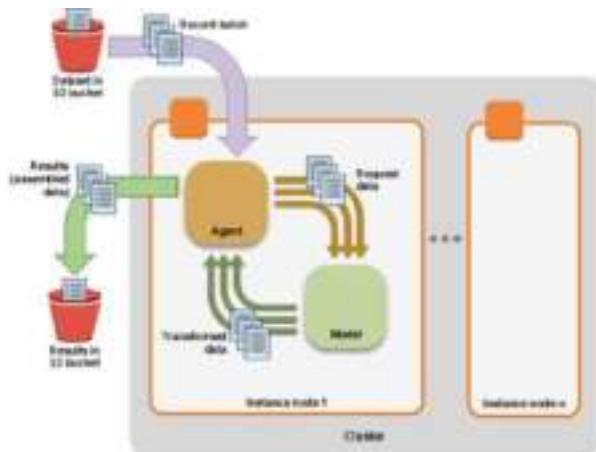
To get inferences for an entire dataset, use batch transform. With batch transform, you create a batch transform job using a trained model and the dataset, which must be stored in Amazon S3. Amazon SageMaker saves the inferences in an S3 bucket that you specify when you create the batch transform job. Batch transform manages all of the compute resources required to get inferences. This includes launching instances and deleting them after the batch transform job has completed. Batch transform manages interactions between the data and the model with an object within the instance node called an agent.

Use batch transform when you:

- Want to get inferences for an entire dataset and index them to serve inferences in real time
- Don't need a persistent endpoint that applications (for example, web or mobile apps) can call to get inferences
- Don't need the subsecond latency that SageMaker hosted endpoints provide

You can also use batch transform to preprocess your data before using it to train a new model or generate inferences.

The following diagram shows the workflow of a batch transform job:



To perform a batch transform, create a batch transform job using either the SageMaker console or the API. Provide the following:

- The path to the S3 bucket where you've stored the data that you want to transform.
- The compute resources that you want SageMaker to use for the transform job. *Compute resources* are machine learning (ML) compute instances that are managed by SageMaker.
- The path to the S3 bucket where you want to store the output of the job.
- The name of the SageMaker model that you want to use to create inferences. You must use a model that you have already created either with the [CreateModel](#) operation or the console.

The following is an example of what a dataset file might look like.

```
An example of input file content:  
AttributeM      Record1-Attribute1, Record1-Attribute2, Record1-Attribute3, ..., Record1-  
AttributeM      Record2-Attribute1, Record2-Attribute2, Record2-Attribute3, ..., Record2-  
AttributeM      Record3-Attribute1, Record3-Attribute2, Record3-Attribute3, ..., Record3-  
AttributeM      ...  
AttributeM      RecordN-Attribute1, RecordN-Attribute2, RecordN-Attribute3, ..., RecordN-
```

A record is a single input data unit. For information about how to delimit records for batch transform jobs, see [SplitType](#).

For an example of how to use batch transform, see [\(Optional\) Make Prediction with Batch Transform \(p. 64\)](#).

## Validate a Machine Learning Model

After training a model, evaluate it to determine whether its performance and accuracy enable you to achieve your business goals. You might generate multiple models using different methods and evaluate each. For example, you could apply different business rules for each model, and then apply various measures to determine each model's suitability. You might consider whether your model needs to be more sensitive than specific (or vice versa).

You can evaluate your model using historical data (offline) or live data:

- **Offline testing**—Use historical, not live, data to send requests to the model for inferences.

Deploy your trained model to an alpha endpoint, and use historical data to send inference requests to it. To send the requests, use a Jupyter notebook in your Amazon SageMaker notebook instance and either the Amazon SDK for Python (Boto) or the high-level Python library provided by SageMaker.

- **Online testing with live data**—SageMaker supports A/B testing for models in production by using production variants. Production variants are models that use the same inference code and are deployed on the same SageMaker endpoint. You configure the production variants so that a small portion of the live traffic goes to the model that you want to validate. For example, you might choose to send 10% of the traffic to a model variant for evaluation. After you are satisfied with the model's performance, you can route 100% traffic to the updated model. For an example of testing models in production, see [Test models in production \(p. 1947\)](#).

For more information, see articles and books about how to evaluate models, for example, [Evaluating Machine Learning Models](#).

Options for offline model evaluation include:

- **Validating using a holdout set**—Machine learning practitioners often set aside a part of the data as a "holdout set." They don't use this data for model training.

With this approach, you evaluate how well your model provides inferences on the holdout set. You then assess how effectively the model generalizes what it learned in the initial training, as opposed to using model memory. This approach to validation gives you an idea of how often the model is able to infer the correct answer.

In some ways, this approach is similar to teaching elementary school students. First, you provide them with a set of examples to learn, and then test their ability to generalize from their learning. With homework and tests, you pose problems that were not included in the initial learning and determine whether they are able to generalize effectively. Students with perfect memories could memorize the problems, instead of learning the rules.

Typically, the holdout dataset is of 20-30% of the training data.

- **k-fold validation**—In this validation approach, you split the example dataset into  $k$  parts. You treat each of these parts as a holdout set for  $k$  training runs, and use the other  $k-1$  parts as the training set for that run. You produce  $k$  models using a similar process, and aggregate the models to generate your final model. The value  $k$  is typically in the range of 5-10.

## Monitoring a Model in Production

After you deploy a model into your production environment, use Amazon SageMaker model monitor to continuously monitor the quality of your machine learning models in real time. Amazon SageMaker model monitor enables you to set up an automated alert triggering system when there are deviations in the model quality, such as data drift and anomalies. Amazon CloudWatch Logs collects log files of monitoring the model status and notifies when the quality of your model hits certain thresholds that you preset. CloudWatch stores the log files to an Amazon S3 bucket you specify. Early and pro-active detection of model deviations through Amazon model monitor products enables you to take prompt actions to maintain and improve the quality of your deployed model.

For more information about SageMaker model monitoring products, see [Amazon SageMaker Model Monitor \(p. 1956\)](#).

To start your machine learning journey with SageMaker, sign up for an Amazon account at [Set Up SageMaker](#).

## Use Machine Learning Frameworks, Python, and R with Amazon SageMaker

You can use Python and R natively in Amazon SageMaker notebook kernels. There are also kernels that support specific frameworks. A very popular way to get started with SageMaker is to use the [Amazon SageMaker Python SDK](#). It provides open source Python APIs and containers that make it easy to train and deploy models in SageMaker, as well as examples for use with several different machine learning and deep learning frameworks.

For information about using specific frameworks or how to use R in SageMaker, see the following topics.

Languages SDKs and user guides:

- [Amazon SageMaker Python SDK](#)
- [R \(p. 28\)](#)
- [API Reference Guide for Amazon SageMaker \(p. 2547\)](#)

Machine learning and deep learning frameworks guides:

- [Apache MXNet \(p. 16\)](#)
- [Apache Spark \(p. 17\)](#)
- [Chainer \(p. 25\)](#)
- [Hugging Face \(p. 26\)](#)
- [PyTorch \(p. 27\)](#)
- [Scikit-learn \(p. 30\)](#)
- [SparkML Serving \(p. 31\)](#)
- [TensorFlow \(p. 31\)](#)

## Use Apache MXNet with Amazon SageMaker

You can use SageMaker to train and deploy a model using custom MXNet code. The [Amazon SageMaker Python SDK](#) MXNet estimators and models and the SageMaker open-source MXNet container make writing a MXNet script and running it in SageMaker easier.

### What do you want to do?

I want to train a custom MXNet model in SageMaker.

For a sample Jupyter notebook, see the [MXNet example notebooks](#) in the Amazon SageMaker Examples GitHub repository.

For documentation, see [Train a Model with MXNet](#).

I have an MXNet model that I trained in SageMaker, and I want to deploy it to a hosted endpoint.

For more information, see [Deploy MXNet models](#).

I have an MXNet model that I trained outside of SageMaker, and I want to deploy it to a SageMaker endpoint

For more information, see [Deploy Endpoints from Model Data](#).

I want to see the API documentation for [Amazon SageMaker Python SDK MXNet classes](#).

For more information, see [MXNet Classes](#).

I want to find the SageMaker MXNet container repository.

For more information, see [SageMaker MXNet Container GitHub repository](#).

I want to find information about MXNet versions supported by Amazon Deep Learning Containers.

For more information, see [Available Deep Learning Container Images](#).

For general information about writing MXNet script mode training scripts and using MXNet script mode estimators and models with SageMaker, see [Using MXNet with the SageMaker Python SDK](#).

## Use Apache Spark with Amazon SageMaker

This section provides information for developers who want to use Apache Spark for preprocessing data and Amazon SageMaker for model training and hosting. For information about supported versions of Apache Spark, see the [Getting SageMaker Spark](#) page in the SageMaker Spark GitHub repository.

SageMaker provides an Apache Spark library, in both Python and Scala, that you can use to easily train models in SageMaker using `org.apache.spark.sql.DataFrame` data frames in your Spark clusters. After model training, you can also host the model using SageMaker hosting services.

The SageMaker Spark library, `com.amazonaws.services.sagemaker.sparksdk`, provides the following classes, among others:

- `SageMakerEstimator`—Extends the `org.apache.spark.ml.Estimator` interface. You can use this estimator for model training in SageMaker.
- `KMeansSageMakerEstimator`, `PCASageMakerEstimator`, and `XGBoostSageMakerEstimator`—Extend the `SageMakerEstimator` class.
- `SageMakerModel`—Extends the `org.apache.spark.ml.Model` class. You can use this `SageMakerModel` for model hosting and obtaining inferences in SageMaker.

## Download the SageMaker Spark Library

You have the following options for downloading the Spark library provided by SageMaker:

- You can download the source code for both PySpark and Scala libraries from the [SageMaker Spark GitHub repository](#).
- For the Python Spark library, you have the following additional options:
  - Use pip install:

```
$ pip install sagemaker_pyspark
```

- In a notebook instance, create a new notebook that uses either the `Sparkmagic (PySpark)` or the `Sparkmagic (PySpark3)` kernel and connect to a remote Amazon EMR cluster.

### Note

The EMR cluster must be configured with an IAM role that has the `AmazonSageMakerFullAccess` policy attached. For information about configuring roles

for an EMR cluster, see [Configure IAM Roles for Amazon EMR Permissions to Amazon Services](#) in the *Amazon EMR Management Guide*.

- You can get the Scala library from Maven. Add the Spark library to your project by adding the following dependency to your `pom.xml` file:

```
<dependency>
  <groupId>com.amazonaws</groupId>
  <artifactId>sagemaker-spark_2.11</artifactId>
  <version>spark_2.2.0-1.0</version>
</dependency>
```

## Integrate Your Apache Spark Application with SageMaker

The following is high-level summary of the steps for integrating your Apache Spark application with SageMaker.

1. Continue data preprocessing using the Apache Spark library that you are familiar with. Your dataset remains a `DataFrame` in your Spark cluster. Load your data into a `DataFrame` and preprocess it so that you have a `features` column with `org.apache.spark.ml.linalg.Vector` of `Doubles`, and an optional `label` column with values of `Double` type.
2. Use the estimator in the SageMaker Spark library to train your model. For example, if you choose the k-means algorithm provided by SageMaker for model training, you call the `KMeansSageMakerEstimator.fit` method.

Provide your `DataFrame` as input. The estimator returns a `SageMakerModel` object.

**Note**

`SageMakerModel` extends the `org.apache.spark.ml.Model`.

The `fit` method does the following:

- a. Converts the input `DataFrame` to the protobuf format by selecting the `features` and `label` columns from the input `DataFrame` and uploading the protobuf data to an Amazon S3 bucket. The protobuf format is efficient for model training in SageMaker.
- b. Starts model training in SageMaker by sending a SageMaker [CreateTrainingJob](#) request. After model training has completed, SageMaker saves the model artifacts to an S3 bucket.

SageMaker assumes the IAM role that you specified for model training to perform tasks on your behalf. For example, it uses the role to read training data from an S3 bucket and to write model artifacts to a bucket.

- c. Creates and returns a `SageMakerModel` object. The constructor does the following tasks, which are related to deploying your model to SageMaker.
    - i. Sends a [CreateModel](#) request to SageMaker.
    - ii. Sends a [CreateEndpointConfig](#) request to SageMaker.
    - iii. Sends a [CreateEndpoint](#) request to SageMaker, which then launches the specified resources, and hosts the model on them.
3. You can get inferences from your model hosted in SageMaker with the `SageMakerModel.transform`.

Provide an input `DataFrame` with features as input. The `transform` method transforms it to a `DataFrame` containing inferences. Internally, the `transform` method sends a request to the [InvokeEndpoint](#) SageMaker API to get inferences. The `transform` method appends the inferences to the input `DataFrame`.

## Example 1: Use Amazon SageMaker for Training and Inference with Apache Spark

### Topics

- [Use Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache Spark \(p. 23\)](#)
- [Use the SageMakerEstimator in a Spark Pipeline \(p. 24\)](#)

Amazon SageMaker provides an Apache Spark library (in both Python and Scala) that you can use to integrate your Apache Spark applications with SageMaker. For example, you might use Apache Spark for data preprocessing and SageMaker for model training and hosting. For more information, see [Use Apache Spark with Amazon SageMaker \(p. 17\)](#). This section provides example code that uses the Apache Spark Scala library provided by SageMaker to train a model in SageMaker using `DataFrames` in your Spark cluster. The example also hosts the resulting model artifacts using SageMaker hosting services. Specifically, this example does the following:

- Uses the `KMeansSageMakerEstimator` to fit (or train) a model on data

Because the example uses the k-means algorithm provided by SageMaker to train a model, you use the `KMeansSageMakerEstimator`. You train the model using images of handwritten single-digit numbers (from the MNIST dataset). You provide the images as an input `DataFrame`. For your convenience, SageMaker provides this dataset in an S3 bucket.

In response, the estimator returns a `SageMakerModel` object.

- Obtains inferences using the trained `SageMakerModel`

To get inferences from a model hosted in SageMaker, you call the `SageMakerModel.transform` method. You pass a `DataFrame` as input. The method transforms the input `DataFrame` to another `DataFrame` containing inferences obtained from the model.

For a given input image of a handwritten single-digit number, the inference identifies a cluster that the image belongs to. For more information, see [K-Means Algorithm \(p. 1420\)](#).

This is the example code:

```
import org.apache.spark.sql.SparkSession
import com.amazonaws.services.sagemaker.sparksdk.IAMRole
import com.amazonaws.services.sagemaker.sparksdk.algorithms
import com.amazonaws.services.sagemaker.sparksdk.algorithms.KMeansSageMakerEstimator

val spark = SparkSession.builder.getOrCreate

// load mnist data as a dataframe from libsvm
val region = "us-east-1"
val trainingData = spark.read.format("libsvm")
    .option("numFeatures", "784")
    .load(s"s3://sagemaker-sample-data-$region/spark/mnist/train/")
val testData = spark.read.format("libsvm")
    .option("numFeatures", "784")
    .load(s"s3://sagemaker-sample-data-$region/spark/mnist/test/")

val roleArn = "arn:aws:iam::account-id:role/rolename"
```

```

val estimator = new KMeansSageMakerEstimator(
    sagemakerRole = IAMRole(roleArn),
    trainingInstanceType = "ml.p2.xlarge",
    trainingInstanceCount = 1,
    endpointInstanceType = "ml.c4.xlarge",
    endpointInitialInstanceCount = 1)
.setK(10).setFeatureDim(784)

// train
val model = estimator.fit(trainingData)

val transformedData = model.transform(testData)
transformedData.show

```

The code does the following:

- Loads the MNIST dataset from an S3 bucket provided by SageMaker (`awsai-sparksdk-dataset`) into a Spark DataFrame (`mnistTrainingDataFrame`):

```

// Get a Spark session.

val spark = SparkSession.builder.getOrCreate

// load mnist data as a dataframe from libsvm
val region = "us-east-1"
val trainingData = spark.read.format("libsvm")
    .option("numFeatures", "784")
    .load(s"s3://sagemaker-sample-data-$region/spark/mnist/train/")
val testData = spark.read.format("libsvm")
    .option("numFeatures", "784")
    .load(s"s3://sagemaker-sample-data-$region/spark/mnist/test/")

val roleArn = "arn:aws:iam::account-id:role/rolename"
trainingData.show()

```

The `show` method displays the first 20 rows in the data frame:

label	features
5.0	(784,[152,153,154...])
0.0	(784,[127,128,129...])
4.0	(784,[160,161,162...])
1.0	(784,[158,159,160...])
9.0	(784,[208,209,210...])
2.0	(784,[155,156,157...])
1.0	(784,[124,125,126...])
3.0	(784,[151,152,153...])
1.0	(784,[152,153,154...])
4.0	(784,[134,135,161...])
3.0	(784,[123,124,125...])
5.0	(784,[216,217,218...])
3.0	(784,[143,144,145...])
6.0	(784,[72,73,74,99...])
1.0	(784,[151,152,153...])
7.0	(784,[211,212,213...])
2.0	(784,[151,152,153...])
8.0	(784,[159,160,161...])
6.0	(784,[100,101,102...])
9.0	(784,[209,210,211...])

only showing top 20 rows

In each row:

- The `label` column identifies the image's label. For example, if the image of the handwritten number is the digit 5, the label value is 5.
- The `features` column stores a vector (`org.apache.spark.ml.linalg.Vector`) of `Double` values. These are the 784 features of the handwritten number. (Each handwritten number is a 28 x 28-pixel image, making 784 features.)
- Creates a SageMaker estimator (`KMeansSageMakerEstimator`)

The `fit` method of this estimator uses the k-means algorithm provided by SageMaker to train models using an input `DataFrame`. In response, it returns a `SageMakerModel` object that you can use to get inferences.

**Note**

The `KMeansSageMakerEstimator` extends the SageMaker `SageMakerEstimator`, which extends the Apache Spark `Estimator`.

```
val estimator = new KMeansSageMakerEstimator(  
    sagemakerRole = IAMRole(roleArn),  
    trainingInstanceType = "ml.p2.xlarge",  
    trainingInstanceCount = 1,  
    endpointInstanceType = "ml.c4.xlarge",  
    endpointInitialInstanceCount = 1)  
    .setK(10).setFeatureDim(784)
```

The constructor parameters provide information that is used for training a model and deploying it on SageMaker:

- `trainingInstanceType` and `trainingInstanceCount`—Identify the type and number of ML compute instances to use for model training.
- `endpointInstanceType`—Identifies the ML compute instance type to use when hosting the model in SageMaker. By default, one ML compute instance is assumed.
- `endpointInitialInstanceCount`—Identifies the number of ML compute instances initially backing the endpoint hosting the model in SageMaker.
- `sagemakerRole`—SageMaker assumes this IAM role to perform tasks on your behalf. For example, for model training, it reads data from S3 and writes training results (model artifacts) to S3.

**Note**

This example implicitly creates a SageMaker client. To create this client, you must provide your credentials. The API uses these credentials to authenticate requests to SageMaker. For example, it uses the credentials to authenticate requests to create a training job and API calls for deploying the model using SageMaker hosting services.

- After the `KMeansSageMakerEstimator` object has been created, you set the following parameters, are used in model training:
  - The number of clusters that the k-means algorithm should create during model training. You specify 10 clusters, one for each digit, 0 through 9.
  - Identifies that each input image has 784 features (each handwritten number is a 28 x 28-pixel image, making 784 features).
- Calls the estimator `fit` method

```
// train
```

```
val model = estimator.fit(trainingData)
```

You pass the input `DataFrame` as a parameter. The model does all the work of training the model and deploying it to SageMaker. For more information see, [Integrate Your Apache Spark Application with SageMaker \(p. 18\)](#). In response, you get a `SageMakerModel` object, which you can use to get inferences from your model deployed in SageMaker.

You provide only the input `DataFrame`. You don't need to specify the registry path to the k-means algorithm used for model training because the `KMeansSageMakerEstimator` knows it.

- Calls the `SageMakerModel.transform` method to get inferences from the model deployed in SageMaker.

The `transform` method takes a `DataFrame` as input, transforms it, and returns another `DataFrame` containing inferences obtained from the model.

```
val transformedData = model.transform(testData)
transformedData.show
```

For simplicity, we use the same `DataFrame` as input to the `transform` method that we used for model training in this example. The `transform` method does the following:

- Serializes the `features` column in the input `DataFrame` to protobuf and sends it to the SageMaker endpoint for inference.
- Deserializes the protobuf response into the two additional columns (`distance_to_cluster` and `closest_cluster`) in the transformed `DataFrame`.

The `show` method gets inferences to the first 20 rows in the input `DataFrame`:

label	features	distance_to_cluster	closest_cluster
5.0 (784,[152,153,154...	1767.897705078125	4.0	
0.0 (784,[127,128,129...	1392.157470703125	5.0	
4.0 (784,[160,161,162...	1671.5711669921875	9.0	
1.0 (784,[158,159,160...	1182.6082763671875	6.0	
9.0 (784,[208,209,210...	1390.4002685546875	0.0	
2.0 (784,[155,156,157...	1713.988037109375	1.0	
1.0 (784,[124,125,126...	1246.3016357421875	2.0	
3.0 (784,[151,152,153...	1753.229248046875	4.0	
1.0 (784,[152,153,154...	978.8394165039062	2.0	
4.0 (784,[134,135,161...	1623.176513671875	3.0	
3.0 (784,[123,124,125...	1533.863525390625	4.0	
5.0 (784,[216,217,218...	1469.357177734375	6.0	
3.0 (784,[143,144,145...	1736.765869140625	4.0	
6.0 (784,[72,73,74,99...	1473.69384765625	8.0	
1.0 (784,[151,152,153...	944.88720703125	2.0	
7.0 (784,[211,212,213...	1285.9071044921875	3.0	
2.0 (784,[151,152,153...	1635.0125732421875	1.0	
8.0 (784,[159,160,161...	1436.3162841796875	6.0	
6.0 (784,[100,101,102...	1499.7366943359375	7.0	
9.0 (784,[209,210,211...	1364.6319580078125	6.0	

You can interpret the data, as follows:

- A handwritten number with the `label` 5 belongs to cluster 4 (`closest_cluster`).
- A handwritten number with the `label` 0 belongs to cluster 5.
- A handwritten number with the `label` 4 belongs to cluster 9.

- A handwritten number with the label 1 belongs to cluster 6.

For more information on how to run these examples, see <https://github.com/aws/sagemaker-spark/blob/master/README.md> on GitHub.

## Use Custom Algorithms for Model Training and Hosting on Amazon SageMaker with Apache Spark

In [Example 1: Use Amazon SageMaker for Training and Inference with Apache Spark \(p. 19\)](#), you use the `KMeansSageMakerEstimator` because the example uses the k-means algorithm provided by Amazon SageMaker for model training. You might choose to use your own custom algorithm for model training instead. Assuming that you have already created a Docker image, you can create your own `SageMakerEstimator` and specify the Amazon Elastic Container Registry path for your custom image.

The following example shows how to create a `KMeansSageMakerEstimator` from the `SageMakerEstimator`. In the new estimator, you explicitly specify the Docker registry path to your training and inference code images.

```
import com.amazonaws.services.sagemaker.sparksdk.IAMRole
import com.amazonaws.services.sagemaker.sparksdk.SageMakerEstimator
import
com.amazonaws.services.sagemaker.sparksdk.transformation.serializers.ProtobufRequestRowSerializer
import
com.amazonaws.services.sagemaker.sparksdk.transformation.deserializers.KMeansProtobufResponseRowDeseri

val estimator = new SageMakerEstimator(
  trainingImage =
    "811284229777.dkr.ecr.us-east-1.amazonaws.com/kmeans:1",
  modelImage =
    "811284229777.dkr.ecr.us-east-1.amazonaws.com/kmeans:1",
  requestRowSerializer = new ProtobufRequestRowSerializer(),
  responseRowDeserializer = new KMeansProtobufResponseRowDeserializer(),
  hyperParameters = Map("k" -> "10", "feature_dim" -> "784"),
  sagemakerRole = IAMRole(roleArn),
  trainingInstanceType = "ml.p2.xlarge",
  trainingInstanceCount = 1,
  endpointInstanceType = "ml.c4.xlarge",
  endpointInitialInstanceCount = 1,
  trainingSparkDataFormat = "sagemaker")
```

In the code, the parameters in the `SageMakerEstimator` constructor include:

- `trainingImage` —Identifies the Docker registry path to the training image containing your custom code.
- `modelImage` —Identifies the Docker registry path to the image containing inference code.
- `requestRowSerializer` —Implements `com.amazonaws.services.sagemaker.sparksdk.transformation.RequestRowSerializer`.

This parameter serializes rows in the input `DataFrame` to send them to the model hosted in SageMaker for inference.

- `responseRowDeserializer` —Implements

```
com.amazonaws.services.sagemaker.sparksdk.transformation.ResponseRowDeserializer.
```

This parameter deserializes responses from the model, hosted in SageMaker, back into a `DataFrame`.

- `trainingSparkDataFormat` —Specifies the data format that Spark uses when uploading training data from a `DataFrame` to S3. For example, "sagemaker" for protobuf format, "csv" for comma-separated values, and "libsvm" for LibSVM format.

You can implement your own `RequestRowSerializer` and `ResponseRowDeserializer` to serialize and deserialize rows from a data format that your inference code supports, such as `.libsvm` or `.csv`.

## Use the SageMakerEstimator in a Spark Pipeline

You can use `org.apache.spark.ml.Estimator` estimators and `org.apache.spark.ml.Model` models, and `SageMakerEstimator` estimators and `SageMakerModel` models in `org.apache.spark.ml.Pipeline` pipelines, as shown in the following example:

```

import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.PCA
import org.apache.spark.sql.SparkSession
import com.amazonaws.services.sagemaker.sparksdk.IAMRole
import com.amazonaws.services.sagemaker.sparksdk.algorithms
import com.amazonaws.services.sagemaker.sparksdk.algorithms.KMeansSageMakerEstimator

val spark = SparkSession.builder.getOrCreate

// load mnist data as a dataframe from libsvm
val region = "us-east-1"
val trainingData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/train/")
val testData = spark.read.format("libsvm")
  .option("numFeatures", "784")
  .load(s"s3://sagemaker-sample-data-$region/spark/mnist/test/")

// substitute your SageMaker IAM role here
val roleArn = "arn:aws:iam::account-id:role/rolename"

val pcaEstimator = new PCA()
  .setInputCol("features")
  .setOutputCol("projectedFeatures")
  .setK(50)

val kMeansSageMakerEstimator = new KMeansSageMakerEstimator(
  sagemakerRole = IAMRole(integTestingRole),
  requestRowSerializer =
    new ProtobufRequestRowSerializer(featuresColumnName = "projectedFeatures"),
  trainingSparkDataFormatOptions = Map("featuresColumnName" -> "projectedFeatures"),
  trainingInstanceType = "ml.p2.xlarge",
  trainingInstanceCount = 1,
  endpointInstanceType = "ml.c4.xlarge",
  endpointInitialInstanceCount = 1)
  .setK(10).setFeatureDim(50)

val pipeline = new Pipeline().setStages(Array(pcaEstimator, kMeansSageMakerEstimator))

// train
val pipelineModel = pipeline.fit(trainingData)

val transformedData = pipelineModel.transform(testData)
transformedData.show()

```

The parameter `trainingSparkDataFormatOptions` configures Spark to serialize to protobuf the "projectedFeatures" column for model training. Additionally, Spark serializes to protobuf the "label" column by default.

Because we want to make inferences using the "projectedFeatures" column, we pass the column name into the `ProtobufRequestRowSerializer`.

The following example shows a transformed DataFrame:

label	features	projectedFeatures	distance_to_cluster	closest_cluster
5.0 (784,[152,153,154... [880.731433034386...  1500.470703125  0.0				
0.0 (784,[127,128,129... [1768.51722024166...  1142.18359375  4.0				
4.0 (784,[160,161,162... [704.949236329314...  1386.246826171875  9.0				
1.0 (784,[158,159,160... [-42.328192193771...  1277.0736083984375  5.0				
9.0 (784,[208,209,210... [374.043902028333...  1211.00927734375  3.0				
2.0 (784,[155,156,157... [941.267714528850...  1496.157958984375  8.0				
1.0 (784,[124,125,126... [30.2848596410594...  1327.6766357421875  5.0				
3.0 (784,[151,152,153... [1270.14374062052...  1570.7674560546875  0.0				
1.0 (784,[152,153,154... [-112.10792566485...  1037.568359375  5.0				
4.0 (784,[134,135,161... [452.068280676606...  1165.1236572265625  3.0				
3.0 (784,[123,124,125... [610.596447285397...  1325.953369140625  7.0				
5.0 (784,[216,217,218... [142.959601818422...  1353.4930419921875  5.0				
3.0 (784,[143,144,145... [1036.71862533658...  1460.4315185546875  7.0				
6.0 (784,[72,73,74,99... [996.740157435754...  1159.8631591796875  2.0				
1.0 (784,[151,152,153... [-107.26076167417...  960.963623046875  5.0				
7.0 (784,[211,212,213... [619.771820430940...  1245.13623046875  6.0				
2.0 (784,[151,152,153... [850.152101817161...  1304.437744140625  8.0				
8.0 (784,[159,160,161... [370.041887230547...  1192.4781494140625  0.0				
6.0 (784,[100,101,102... [546.674328209335...  1277.0908203125  2.0				
9.0 (784,[209,210,211... [-29.259112927426...  1245.8182373046875  6.0				

## SDK examples: Use Amazon SageMaker with Apache Spark

The following list is a subset of available examples. Visit the [examples website](#) to see more.

- [sagemaker-spark](#): a Spark library for SageMaker
- [SageMaker PySpark K-Means Clustering MNIST Example](#)
- [Distributed Data Processing using Apache Spark and SageMaker Processing](#)
- [Feature processing with Spark, training with XGBoost and deploying as Inference Pipeline](#)

### Note

To run the notebooks on a notebook instance, see [Example Notebooks \(p. 131\)](#). To run the notebooks on Studio, see [Create or Open an Amazon SageMaker Studio Notebook \(p. 79\)](#).

## Use Chainer with Amazon SageMaker

You can use SageMaker to train and deploy a model using custom Chainer code. The SageMaker Python SDK Chainer estimators and models and the SageMaker open-source Chainer container make writing a Chainer script and running it in SageMaker easier.

### What do you want to do?

I want to train a custom Chainer model in SageMaker.

For a sample Jupyter notebook, see the [Chainer example notebooks](#) in the Amazon SageMaker Examples GitHub repository.

For documentation, see [Train a Model with Chainer](#).

I have a Chainer model that I trained in SageMaker, and I want to deploy it to a hosted endpoint.

For more information, see [Deploy Chainer models](#).

I have a Chainer model that I trained outside of SageMaker, and I want to deploy it to a SageMaker endpoint

For more information, see [Deploy Endpoints from Model Data](#).

I want to see the API documentation for [Amazon SageMaker Python SDK Chainer classes](#).

For more information, see [Chainer Classes](#).

I want to find information about SageMaker Chainer containers.

For more information, see the [SageMaker Chainer Container GitHub repository](#).

For information about supported Chainer versions, and for general information about writing Chainer training scripts and using Chainer estimators and models with SageMaker, see [Using Chainer with the SageMaker Python SDK](#).

## Use Hugging Face with Amazon SageMaker

Amazon SageMaker enables customers to train, fine-tune, and run inference using Hugging Face models for Natural Language Processing (NLP) on SageMaker. You can use any of the thousands of models available in Hugging Face and fine-tune them for your specific use case with additional training. With SageMaker, you can use standard training or take advantage of [SageMaker Distributed Data and Model Parallel training](#). You can also debug your training jobs using [Amazon SageMaker Debugger](#). As with other SageMaker training jobs using custom code, you can capture your own metrics by passing a metrics definition to the SageMaker Python SDK as shown in [Defining Training Metrics \(SageMaker Python SDK\)](#). The captured metrics are then accessible via [CloudWatch](#) and as a Pandas DataFrame via the [TrainingJobAnalytics](#) method. Once your model is trained and fine-tuned, you can use it like any other model to run inference jobs.

This functionality is available through the development of a Hugging Face [Deep Learning Container](#). These containers include Hugging Face Transformers, Tokenizers and the Datasets library, which allows you to use these resources for your training jobs. For a list of the available DLC images, see [Available Deep Learning Containers Images](#).

To use the Hugging Face Deep Learning Container with the SageMaker Python SDK, see the [Hugging Face SageMaker Estimator](#). With the Hugging Face Estimator, you can use the Hugging Face resources as you would any other SageMaker estimator.

For more information on Hugging Face and the models available in it, see the [Hugging Face documentation](#).

## How to Use the Hugging Face Estimator

You can implement the Hugging Face Estimator for training jobs using the SageMaker Python SDK. The SageMaker Python SDK is an open source library for training and deploying machine learning models on SageMaker. For more information on the Hugging Face Estimator, see the [SageMaker Python SDK documentation](#).

With the SageMaker Python SDK, you can run training jobs using the Hugging Face Estimator in the following environments:

- **SageMaker Studio:** Amazon SageMaker Studio is the first fully integrated development environment (IDE) for machine learning (ML). SageMaker Studio provides a single, web-based visual interface where you can perform all ML development steps required to prepare, build, train and tune, deploy and manage models. For information on using Jupyter Notebooks in Studio, see [Use Amazon SageMaker Studio Notebooks](#).
- **SageMaker Notebook Instances:** An Amazon SageMaker notebook instance is a machine learning (ML) compute instance running the Jupyter Notebook App. This app lets you run Jupyter Notebooks

in your notebook instance to prepare and process data, write code to train models, deploy models to SageMaker hosting, and test or validate your models without SageMaker Studio features like Debugger, Model Monitoring, and a web-based IDE.

- Locally: If you have connectivity to Amazon and have appropriate SageMaker permissions, you can use the SageMaker Python SDK locally to launch remote training and inference jobs for Hugging Face in SageMaker on Amazon.

### What do you want to do?

The following Jupyter Notebooks illustrate how to use the Hugging Face Estimator with SageMaker in various use cases.

I want to train a text classification model using Hugging Face in SageMaker with PyTorch.

For a sample Jupyter Notebook, see the [PyTorch Getting Started Demo](#).

I want to train a text classification model using Hugging Face in SageMaker with TensorFlow.

For a sample Jupyter Notebook, see the [TensorFlow Getting Started example](#).

I want to run distributed training with data parallelism using Hugging Face and SageMaker Distributed.

For a sample Jupyter Notebook, see the [Distributed Training example](#).

I want to run distributed training with model parallelism using Hugging Face and SageMaker Distributed.

For a sample Jupyter Notebook, see the [Model Parallelism example](#).

I want to use a spot instance to train a model using Hugging Face in SageMaker.

For a sample Jupyter Notebook, see the [Spot Instances example](#).

I want to capture custom metrics and use SageMaker Checkpointing when training a text classification model using Hugging Face in SageMaker.

For a sample Jupyter Notebook, see the [Training with Custom Metrics example](#).

I want to train a distributed question-answering TensorFlow model using Hugging Face in SageMaker.

For a sample Jupyter Notebook, see the [Distributed TensorFlow Training example](#).

## Use PyTorch with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom PyTorch code. The SageMaker Python SDK PyTorch estimators and models and the SageMaker open-source PyTorch container make writing a PyTorch script and running it in SageMaker easier.

### What do you want to do?

I want to train a custom PyTorch model in SageMaker.

For a sample Jupyter notebook, see the [PyTorch example notebook](#) in the Amazon SageMaker Examples GitHub repository.

For documentation, see [Train a Model with PyTorch](#).

I have a PyTorch model that I trained in SageMaker, and I want to deploy it to a hosted endpoint.

For more information, see [Deploy PyTorch models](#).

I have a PyTorch model that I trained outside of SageMaker, and I want to deploy it to a SageMaker endpoint

For more information, see [Deploy Endpoints from Model Data](#).

I want to see the API documentation for [Amazon SageMaker Python SDK PyTorch classes](#).

For more information, see [PyTorch Classes](#).

I want to find the SageMaker PyTorch container repository.

For more information, see [SageMaker PyTorch Container GitHub repository](#).

I want to find information about PyTorch versions supported by Amazon Deep Learning Containers.

For more information, see [Available Deep Learning Container Images](#).

For general information about writing PyTorch training scripts and using PyTorch estimators and models with SageMaker, see [Using PyTorch with the SageMaker Python SDK](#).

## R User Guide to Amazon SageMaker

This document will walk you through ways of leveraging Amazon SageMaker features using R. This guide introduces SageMaker's built-in R kernel, how to get started with R on SageMaker, and finally several example notebooks.

The examples are organized in three levels, Beginner, Intermediate, and Advanced. They start from [Getting Started with R on SageMaker](#), continue to end-to-end machine learning with R on SageMaker, and then finish with more advanced topics such as SageMaker Processing with R script, and Bring-Your-Own (BYO) R algorithm to SageMaker.

For information on how to bring your own custom R image to Studio, see [Bring your own SageMaker image \(p. 95\)](#). For a similar blog article, see [Bringing your own R environment to Amazon SageMaker Studio](#).

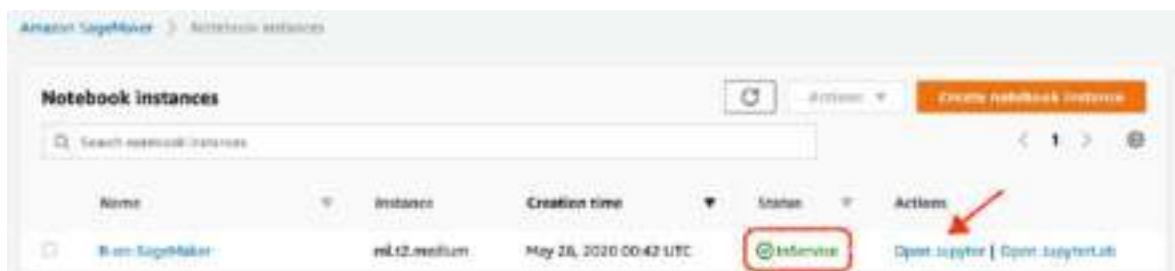
## R Kernel in SageMaker

SageMaker notebook instances support R using a pre-installed R kernel. Also, the R kernel has the reticulate library, an R to Python interface, so you can use the features of SageMaker Python SDK from within an R script.

- [reticulatelibrary](#): provides an R interface to the [Amazon SageMaker Python SDK](#). The reticulate package translates between R and Python objects.

## Get Started with R in SageMaker

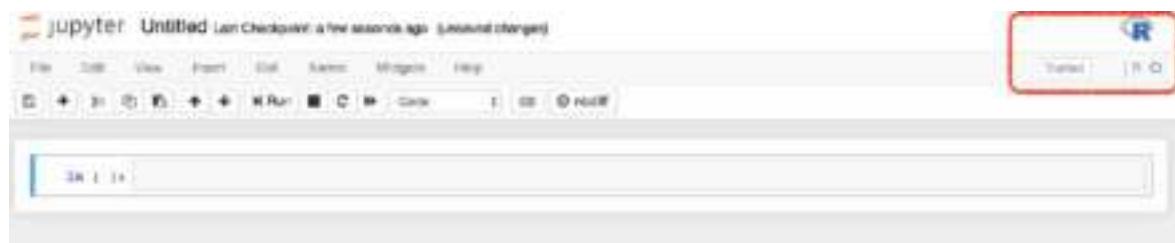
- [Create a Notebook Instance](#) using the t2.medium instance type and default storage size. You can pick a faster instance and more storage if you plan to continue using the instance for more advanced examples, or create a bigger instance later.
- Wait until the status of the notebook is In Service, and then click Open Jupyter.



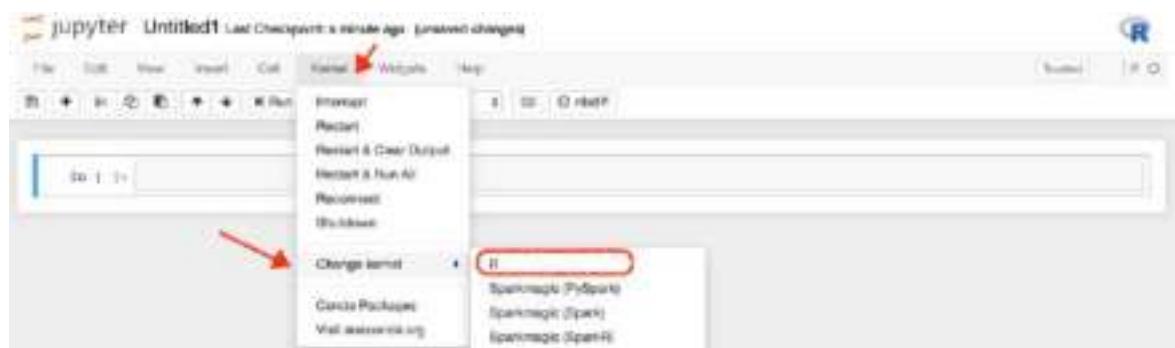
- Create a new notebook with R kernel from the list of available environments.



- When the new notebook is created, you should see an R logo in the upper right corner of the notebook environment, and also R as the kernel under that logo. This indicates that SageMaker has successfully launched the R kernel for this notebook.



- Alternatively, when you are in a Jupyter notebook, you can use Kernel menu, and then select R from Change Kernel option.



## Example Notebooks

### Prerequisites

[Getting Started with R on SageMaker](#): This sample notebook describes how you can develop R scripts using Amazon SageMaker's R kernel. In this notebook you set up your SageMaker environment and permissions, download the [abalone dataset](#) from the [UCI Machine Learning Repository](#), do some basic processing and visualization on the data, then save the data as .csv format to S3.

### Beginner Level

[SageMaker Batch Transform using R Kernel](#): This sample Notebook describes how to conduct a batch transform job using SageMaker's Transformer API and the [XGBoost algorithm](#). The notebook also uses the Abalone dataset.

## Intermediate Level

[Hyperparameter Optimization for XGBoost in R](#): This sample notebook extends the previous beginner notebooks that use the abalone dataset and XGBoost. It describes how to do model tuning with [hyperparameter optimization](#). You will also learn how to use batch transform for batching predictions, as well as how to create a model endpoint to make real-time predictions.

[Amazon SageMaker Processing with R](#): [SageMaker Processing](#) lets you preprocess, post-process and run model evaluation workloads. This example shows you how to create an R script to orchestrate a Processing job.

## Advanced Level

[Train and Deploy Your Own R Algorithm in SageMaker](#): Do you already have an R algorithm, and you want to bring it into SageMaker to tune, train, or deploy it? This example walks you through how to customize SageMaker containers with custom R packages, all the way to using a hosted endpoint for inference on your R-origin model.

# Use Scikit-learn with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom Scikit-learn code. The SageMaker Python SDK Scikit-learn estimators and models and the SageMaker open-source Scikit-learn container make writing a Scikit-learn script and running it in SageMaker easier.

## What do you want to do?

I want to use Scikit-learn for data processing, feature engineering, or model evaluation in SageMaker.

For a sample Jupyter notebook, see [https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker\\_processing/scikit\\_learn\\_data\\_processing\\_and\\_model\\_evaluation](https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker_processing/scikit_learn_data_processing_and_model_evaluation).

For documentation, see [ReadTheDocs](#).

I want to train a custom Scikit-learn model in SageMaker.

For a sample Jupyter notebook, see [https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/scikit\\_learn\\_iris](https://github.com/awslabs/amazon-sagemaker-examples/tree/master/sagemaker-python-sdk/scikit_learn_iris).

For documentation, see [Train a Model with Scikit-learn](#).

I have a Scikit-learn model that I trained in SageMaker, and I want to deploy it to a hosted endpoint.

For more information, see [Deploy Scikit-learn models](#).

I have a Scikit-learn model that I trained outside of SageMaker, and I want to deploy it to a SageMaker endpoint

For more information, see [Deploy Endpoints from Model Data](#).

I want to see the API documentation for [Amazon SageMaker Python SDK Scikit-learn classes](#).

For more information, see [Scikit-learn Classes](#).

I want to see information about SageMaker Scikit-learn containers.

For more information, see [SageMaker Scikit-learn Container GitHub repository](#).

For general information about writing Scikit-learn training scripts and using Scikit-learn estimators and models with SageMaker, see [Using Scikit-learn with the SageMaker Python SDK](#).

Scikit-learn versions supported by the Amazon SageMaker Scikit-learn container: 0.20.0, 0.23-1.

## Use SparkML Serving with Amazon SageMaker

The [Amazon SageMaker Python SDK](#) SparkML Serving model and predictor and the Amazon SageMaker open-source SparkML Serving container support deploying Apache Spark ML pipelines serialized with MLeap in SageMaker to get inferences.

For information about using the SparkML Serving container to deploy models to SageMaker, see [SageMaker Spark ML Container GitHub repository](#). For information about the [Amazon SageMaker Python SDK](#) SparkML Serving model and predictors, see the [SparkML Serving Model and Predictor API documentation](#).

## Use TensorFlow with Amazon SageMaker

You can use Amazon SageMaker to train and deploy a model using custom TensorFlow code. The SageMaker Python SDK TensorFlow estimators and models and the SageMaker open-source TensorFlow containers make writing a TensorFlow script and running it in SageMaker easier.

### Use TensorFlow Version 1.11 and Later

For TensorFlow versions 1.11 and later, the [Amazon SageMaker Python SDK](#) supports script mode training scripts.

#### What do you want to do?

I want to train a custom TensorFlow model in SageMaker.

For a sample Jupyter notebook, see [TensorFlow script mode training and serving](#).

For documentation, see [Train a Model with TensorFlow](#).

I have a TensorFlow model that I trained in SageMaker, and I want to deploy it to a hosted endpoint.

For more information, see [Deploy TensorFlow Serving models](#).

I have a TensorFlow model that I trained outside of SageMaker, and I want to deploy it to a SageMaker endpoint

For more information, see [Deploying directly from model artifacts](#).

I want to see the API documentation for [Amazon SageMaker Python SDK TensorFlow classes](#).

For more information, see [TensorFlow Estimator](#).

I want to find the SageMaker TensorFlow container repository.

For more information, see [SageMaker TensorFlow Container GitHub repository](#).

I want to find information about TensorFlow versions supported by Amazon Deep Learning Containers.

For more information, see [Available Deep Learning Container Images](#).

For general information about writing TensorFlow script mode training scripts and using TensorFlow script mode estimators and models with SageMaker, see [Using TensorFlow with the SageMaker Python SDK](#).

### Use TensorFlow Legacy Mode for Versions 1.11 and Earlier

The [Amazon SageMaker Python SDK](#) provides a legacy mode that supports TensorFlow versions 1.11 and earlier. Use legacy mode TensorFlow training scripts to run TensorFlow jobs in SageMaker if:

- You have existing legacy mode scripts that you do not want to convert to script mode.
- You want to use a TensorFlow version earlier than 1.11.

For information about writing legacy mode TensorFlow scripts to use with the SageMaker Python SDK, see [TensorFlow SageMaker Estimators and Models](#).

## Supported Regions and Quotas

For the Amazon Regions supported by Amazon SageMaker and the Amazon Elastic Compute Cloud (Amazon EC2) instance types that are available in each Region, see [Amazon SageMaker Pricing](#).

For a list of the SageMaker service endpoints for each Region and the SageMaker service quotas for each instance type, see [Amazon SageMaker endpoints and quotas](#).

Amazon SageMaker Studio is available in all the Amazon Regions supported by Amazon SageMaker except the Amazon GovCloud (US) Regions. In the supported Regions, Studio is available in the same Availability Zones as notebook instances.

Amazon SageMaker Pipelines is available in all the Amazon Regions supported by Amazon except the Amazon GovCloud (US) Regions. SageMaker Projects is available in the Amazon regions where CodePipeline is available. For more information about CodePipeline region availability, see the [Amazon Regional Services List](#).

For the Availability Zones supported by SageMaker, see [Availability Zones \(p. 2546\)](#).

### About service quotas

Depending on your activities and resource usage over time, your SageMaker quotas might be different from the default SageMaker quotas listed on [Amazon SageMaker endpoints and quotas](#) in the *Amazon General Reference*. If you encounter error messages that you've exceeded your quota and you need to scale up your SageMaker resources, follow the steps in the [Request a service quota increase for SageMaker resources \(p. 32\)](#) procedure on this page to request a quota increase from Amazon Support. The SageMaker service team reviews and evaluates requests on case-by-case basis. For additional information, see [How do I resolve the ResourceLimitExceeded error in Amazon SageMaker?](#) in the Amazon Web Services Support Knowledge Center and [Amazon Service Quotas](#) in the *Amazon General Reference*.

## Request a service quota increase for SageMaker resources

Follow the steps in this procedure to request a quota (also known as *limit*) increase for SageMaker resources.

### To request a service quota increase

1. Sign in to the Amazon Web Services Support console at [Amazon Web Services Support Center](#).
2. In the left navigation pane, choose **Your support cases**, and then choose **Create case**.
3. For **Limit type** in the **Case details** section, choose the specific SageMaker component.
4. For **Request 1** in the **Requests** section, choose **Region**, **Resource Type**, and **Limit** for a resource which you want to increase the quota.
5. For **New limit value**, type an integer value for a new quota.
6. For **Case description**, include the following items:

- a. The error message showing the current quota. For example, the error message should look like the following:

```
ResourceLimitExceeded: An error occurred (ResourceLimitExceeded) when calling the CreateTrainingJob operation: The account-level service limit 'ml.p3dn.24xlarge for training job usage' is 0 Instances, with current utilization of 0 Instances and a request delta of 1 Instances.  
Please contact Amazon support to request an increase for this limit.
```

- b. A brief description of your use case that requires the service quota increase.
7. If you want to add more quota increase requests for other resources, choose **Add another request** and repeat the previous steps.
8. Choose **Submit** to finish the request.

# Get Started with Amazon SageMaker

This video shows you how to setup and use SageMaker Studio. (Length: 19:14)

Before you can use Amazon SageMaker, you must sign up for an Amazon account, create an IAM admin user, and onboard to Amazon SageMaker Studio.

After you complete these tasks, try out the Get Started guides. The guides walk you through training your first model using SageMaker Studio, or the SageMaker console and the SageMaker API.

## Topics

- [Set Up Amazon SageMaker \(p. 34\)](#)
- [Onboard to Amazon SageMaker Studio \(p. 35\)](#)
- [SageMaker JumpStart \(p. 40\)](#)
- [Get Started with Amazon SageMaker Notebook Instances \(p. 51\)](#)

## Set Up Amazon SageMaker

In this section, you sign up for an Amazon account, create an IAM admin user, and onboard to Amazon SageMaker Studio.

If you're new to SageMaker, we recommend that you read [How Amazon SageMaker Works \(p. 3\)](#).

## Topics

- [Create an Amazon Account \(p. 34\)](#)
- [Create an IAM Administrator User and Group \(p. 34\)](#)

## Create an Amazon Account

In this section, you sign up for an Amazon account. If you already have an Amazon account, skip this step.

When you sign up for Amazon Web Services (Amazon), your Amazon account is automatically signed up for all Amazon services, including SageMaker. You are charged only for the services that you use.

### To create an Amazon account

1. Open <https://portal.amazonaws.cn/billing/signup>.
2. Follow the online instructions.

Part of the sign-up procedure involves receiving a phone call and entering a verification code on the phone keypad.

Write down your Amazon account ID because you'll need it for the next task.

## Create an IAM Administrator User and Group

When you create an Amazon account, you get a single sign-in identity that has complete access to all of the Amazon services and resources in the account. This identity is called the Amazon account *root*

*user*. Signing in to the Amazon console using the email address and password that you used to create the account gives you complete access to all of the Amazon resources in your account.

We strongly recommend that you *not* use the root user for everyday tasks, even the administrative ones. Instead, adhere to the [Create Individual IAM Users](#), an Amazon Identity and Access Management (IAM) administrator user. Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

#### To create an administrator user

- Create an administrator user in your Amazon account. For instructions, see [Creating Your First IAM User and Administrators Group](#) in the *IAM User Guide*.

##### Note

We assume that you use administrator user credentials for the exercises and procedures in this guide. If you choose to create and use another IAM user, grant that user minimum permissions. For more information, see [Authenticating with Identities \(p. 2418\)](#).

## Onboard to Amazon SageMaker Studio

To use Amazon SageMaker Studio and Amazon SageMaker Studio Notebooks, you must complete the Studio onboarding process using the SageMaker console.

The simplest way to create a Amazon SageMaker Studio account is to follow the **Quick start** procedure. Quick start uses the same default settings as the **Standard setup** procedures. These settings include shareable notebooks and public internet access. For more control, including the option of using Amazon Web Services SSO authentication, use the **Standard setup** procedures.

#### Topics

- [Onboard to Amazon SageMaker Studio Using Quick Start \(p. 35\)](#)
- [Onboard to Amazon SageMaker Studio Using IAM \(p. 36\)](#)
- [Choose a VPC \(p. 38\)](#)
- [Delete an Amazon SageMaker Studio Domain \(p. 39\)](#)

## Onboard to Amazon SageMaker Studio Using Quick Start

This topic describes how to onboard to Amazon SageMaker Studio using the **Quick start** procedure, which uses Amazon Identity and Access Management (IAM) authentication. For information on how to onboard using the standard IAM procedure, see [Onboard Using IAM \(p. 36\)](#).

#### To onboard to Studio using Quick start

1. Open the [SageMaker console](#).
2. Choose **Amazon SageMaker Studio** at the top left of the page.
3. On the SageMaker Studio page, under **Get started**, choose **Quick start**.
4. For **User name**, keep the default name or create a new name. The name can be up to 63 characters. Valid characters: A-Z, a-z, 0-9, and - (hyphen).
5. For **Execution role**, choose an option from the role selector.

If you choose **Enter a custom IAM role ARN**, the role must have at a minimum, an attached trust policy that grants SageMaker permission to assume the role. For more information, see [SageMaker Roles \(p. 2447\)](#).

If you choose **Create a new role**, the **Create an IAM role** dialog opens:

- For **S3 buckets you specify**, specify additional S3 buckets that users of your notebooks can access. If you don't want to add access to more buckets, choose **None**.
  - Choose **Create role**. SageMaker creates a new IAM `AmazonSageMaker-ExecutionPolicy` role with the `AmazonSageMakerFullAccess` policy attached.
6. For **Projects**, see [SageMaker Studio Permissions Required to Use Projects \(p. 2236\)](#). For more control, use the standard setup procedure.
  7. Choose **Submit**.

**Note**

If you receive an error message that you need to create a VPC, see [Choose a VPC \(p. 38\)](#).

On the **SageMaker Studio Control Panel**, under **Studio Summary**, wait for **Status** to change to **Ready**.

When **Status** is **Ready**, the user name that you specified is enabled and chosen. The **Add user** and **Delete user** buttons, and the **Open Studio** link are also enabled.

8. Choose **Open Studio**. The **Amazon SageMaker Studio** loading page displays.

When Studio opens you can start using it.

Now that you've onboarded to SageMaker Studio, use the following steps to access Studio later.

**To access Studio after you onboard**

1. Open the [SageMaker console](#).
2. Choose **Amazon SageMaker Studio** at the top left of the page.
3. On the **Amazon SageMaker Studio Control Panel**, choose your user name and then choose **Open Studio**.

**To add more users**

1. On the **Amazon SageMaker Studio Control Panel**, choose **Add user**.
2. Repeat steps 4 and 5 from the first procedure, "To onboard to Studio using Quick start."
3. Choose **Submit**.

For information about using SageMaker Studio, see [SageMaker Studio \(p. 69\)](#).

## Onboard to Amazon SageMaker Studio Using IAM

This topic describes how to onboard to Amazon SageMaker Studio using the standard setup procedure for Amazon Identity and Access Management (IAM) authentication. To onboard faster using IAM, see [Onboard Using Quick Start \(p. 35\)](#).

**To onboard to Studio using IAM**

1. Open the [SageMaker console](#).
2. Choose **Amazon SageMaker Studio** at the top left of the page.
3. On the **SageMaker Studio** page, under **Get started**, choose **Standard setup**.
4. For **Authentication method**, choose **Amazon Identity and Access Management (IAM)**.
5. Under **Permission**, for **Execution role for all users**, choose an option from the role selector.

If you choose **Enter a custom IAM role ARN**, the role must have at a minimum, an attached trust policy that grants SageMaker permission to assume the role. For more information, see [SageMaker Roles \(p. 2447\)](#).

If you choose **Create a new role**, the **Create an IAM role** dialog opens:

- a. For **S3 buckets you specify**, specify additional S3 buckets that users of your notebooks can access. If you don't want to add access to more buckets, choose **None**.
  - b. Choose **Create role**. SageMaker creates a new IAM `AmazonSageMaker-ExecutionPolicy` role with the [AmazonSageMakerFullAccess](#) policy attached.
6. For **Projects**, see [SageMaker Studio Permissions Required to Use Projects \(p. 2236\)](#).
  7. Under **Network and storage**, specify the following:
    - Your VPC information – For more information, see [Choose a VPC \(p. 38\)](#).
    - (Optional) **Storage encryption key** – SageMaker uses an Amazon managed customer master key (CMK) to encrypt your Amazon Elastic File System (Amazon EFS) and Amazon Elastic Block Store (Amazon EBS) file systems by default. To use a customer managed CMK, enter its key ID or Amazon Resource Name (ARN). For more information, see [Protect Data at Rest Using Encryption \(p. 2413\)](#).
  8. Choose **Submit**.

On the **Amazon SageMaker Studio Control Panel**, under **Studio Summary**, wait for **Status** to change to **Ready** and the **Add user** button to be enabled.

9. Choose **Add user**.
10. On the **Add user** page, keep the default name or create a new name. A name can be up to 63 characters. Valid characters: A-Z, a-z, 0-9, and - (hyphen).  
For **Execution role**, choose an option from the role selector.
11. Choose **Submit**. The **Amazon SageMaker Studio Control Panel** opens with the new user listed. The **Delete user** button and the **Open Studio** link are both enabled.
12. To add more users, repeat steps 7 through 9.

When multiple users are listed, none of the users in the user list is chosen. Choose a user and the **Delete user** button and the **Open Studio** link for that user are enabled.

13. Choose **Open Studio**. The **Amazon SageMaker Studio** loading page displays.

When SageMaker Studio opens, you can start using Studio.

Now that you've onboarded to SageMaker Studio, use the following steps to subsequently access Studio.

### Access Studio after you onboard

1. Open the [SageMaker console](#).
2. Choose **Amazon SageMaker Studio** at the top left of the page.
3. On the **Amazon SageMaker Studio Control Panel**, choose your user name and then choose **Open Studio**.

For information about using SageMaker Studio, see [SageMaker Studio \(p. 69\)](#).

## Choose a VPC

This topic provides detailed information on choosing an Amazon Virtual Private Cloud (VPC) when you onboard to Amazon SageMaker Studio. For more information on onboarding to SageMaker Studio, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).

By default, SageMaker Studio uses two VPCs. One VPC is managed by Amazon SageMaker and provides direct internet access. You specify the other VPC, which provides encrypted traffic between the domain and your Amazon Elastic File System (EFS) volume.

You can change this behavior so that Studio sends all traffic over your specified VPC. When you choose this option, you must provide the subnets, security groups, and interface endpoints that are necessary to communicate with the SageMaker API and SageMaker runtime, and various Amazon services, such as Amazon Simple Storage Service (Amazon S3) and Amazon CloudWatch, that are used by Studio and your Studio notebooks.

When you onboard to Studio, you tell Studio to send all traffic over your VPC by setting the network access type to **VPC only**.

### To specify the VPC information

When you specify the VPC entities (that is, the VPC, subnet, or security group) in the following procedure, one of three options is presented based on the number of entities you have in the current Amazon Region. The behavior is as follows:

- One entity – Studio uses that entity. This can't be changed.
- Multiple entities – You must choose the entities from the dropdown list.
- No entities – You must create one or more entities in order to use Studio. Choose **Create <entity>** to open the VPC console in a new browser tab. After you create the entities, return to the Studio **Get started** page to continue the onboarding process.

This procedure is part of the SageMaker Studio onboarding process when you choose **Standard setup**. Your VPC information is specified under the **Network** section.

1. Choose the VPC.
2. Choose one or more subnets. If you don't choose any subnets, SageMaker uses all the subnets in the VPC.
3. Select the network access type.
  - **Public internet only** – Non-EFS traffic goes through a SageMaker managed VPC, which allows internet access. Traffic between the domain and your Amazon EFS volume is through the specified VPC.
  - **VPC only** – All Studio traffic is through the specified VPC and subnets. Internet access is disabled by default.
4. Choose the security groups. If you chose **Public internet only**, this step is optional. If you chose **VPC only**, this step is required.

#### Note

For the maximum number of allowed security groups, see [UserSettings](#).

For VPC requirements in **VPC only** mode, see [Connect SageMaker Studio Notebooks to Resources in a VPC \(p. 2491\)](#).

## Delete an Amazon SageMaker Studio Domain

When you onboard to Amazon SageMaker Studio using IAM authentication, Studio creates a domain for your account. A domain consists of a list of authorized users, configuration settings, and an Amazon Elastic File System (Amazon EFS) volume, which contains data for the users, including notebooks, resources, and artifacts. A user can have multiple applications (apps) which support the reading and execution experience of the user's notebooks, terminals, and consoles. For more information on the EFS volume, see [Manage Your EFS Storage Volume in SageMaker Studio \(p. 116\)](#).

To return Studio to the state it was in before you onboarded, you must delete this domain. You can delete the domain by using the Studio Control Panel, the Amazon Command Line Interface (Amazon CLI), or the SageMaker SDK. When you use the Studio Control Panel to delete the domain, the Amazon EFS volume is detached but not deleted. The same behavior occurs by default when you use the Amazon CLI or the SDK to delete the domain. However, when you use the Amazon CLI or the SDK, you can set the `RetentionPolicy` to `HomeEfsFileSystem=Delete` to delete the EFS volume along with the domain.

To delete a domain, the domain cannot contain any user profiles. To delete a user profile, the profile cannot contain any non-failed apps.

When you delete these resources, the following occurs:

- App – The data (files and notebooks) in a user's home directory is saved. Unsaved notebook data is lost.
- User profile – The user is no longer able to sign in to Studio and loses access to their home directory but the data is not deleted. An admin can retrieve the data from the Amazon EFS volume where it is stored under the user's Amazon account.

### Note

You must have admin permission to delete a domain.

You can only delete an app whose status is `InService`, which is displayed as **Ready** in Studio. An app whose status is `Failed` doesn't need to be deleted to delete the containing domain. In Studio, an attempt to delete an app in the failed state results in an error.

### Topics

- [Delete a SageMaker Studio Domain \(Studio\) \(p. 39\)](#)
- [Delete a SageMaker Studio Domain \(CLI\) \(p. 40\)](#)

## Delete a SageMaker Studio Domain (Studio)

### To delete a domain

1. Open the [SageMaker console](#).
2. Choose **Amazon SageMaker Studio** at the top left of the page to open the **Amazon SageMaker Studio Control Panel**.
3. Repeat the following steps for each user in the **User name** list.
  - a. Choose the user.
  - b. On the **User Details** page, for each non-failed app in the **Apps** list, choose **Delete app**.
  - c. On the **Delete app** dialog, choose **Yes, delete app**, type `delete` in the confirmation field, and then choose **Delete**.
  - d. When the **Status** for all apps show as **Deleted**, choose **Delete user**.

**Important**

When a user is deleted, they lose access to the Amazon EFS volume that contains their data, including notebooks and other artifacts. The data is not deleted and can be accessed by an administrator.

4. When all users are deleted, choose **Delete Studio**.
5. On the **Delete Studio** dialog, choose **Yes, delete Studio**, type *delete* in the confirmation field, and then choose **Delete**.

## Delete a SageMaker Studio Domain (CLI)

### To delete a domain

1. Retrieve the list of domains in your account.

```
aws --region Region sagemaker list-domains
```

2. Retrieve the list of applications for the domain to be deleted.

```
aws --region Region sagemaker list-apps \  
--domain-id-equals DomainId
```

3. Delete each application in the list.

```
aws --region Region sagemaker delete-app \  
--domain-id DomainId \  
--app-name AppName \  
--app-type AppType \  
--user-profile-name UserProfileName
```

4. Retrieve the list of user profiles in the domain.

```
aws --region Region sagemaker list-user-profiles \  
--domain-id-equals DomainId
```

5. Delete each user profile in the list.

```
aws --region Region sagemaker delete-user-profile \  
--domain-id DomainId \  
--user-profile-name UserProfileName
```

6. Delete the domain. To also delete the Amazon EFS volume, specify `HomeEfsFileSystem=Delete`.

```
aws --region Region sagemaker delete-domain \  
--domain-id DomainId \  
--retention-policy HomeEfsFileSystem=Retain
```

## SageMaker JumpStart

**Important**

To use new features with an existing notebook instance or Studio app, you must restart the notebook instance or the Studio app to get the latest updates.

You can use SageMaker JumpStart to learn about SageMaker features and capabilities through curated 1-click solutions, example notebooks, and pretrained models that you can deploy. You can also fine-tune the models and deploy them.

To access JumpStart, you must first launch SageMaker Studio. JumpStart features are not available in SageMaker notebook instances, and you can't access them through SageMaker APIs or the Amazon CLI.

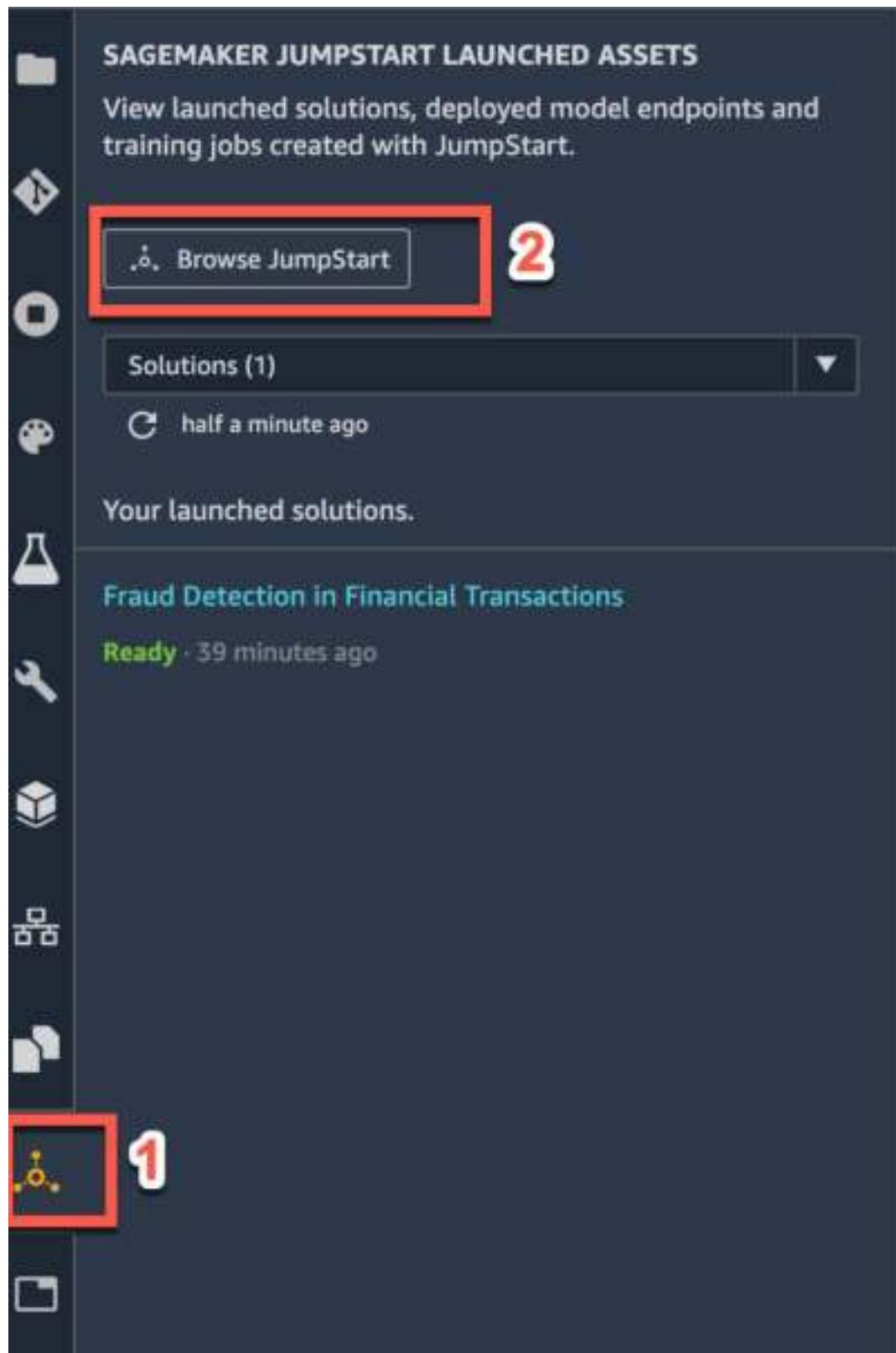
Open JumpStart by using the JumpStart launcher in the **Get Started** section or by choosing the



JumpStart icon ( ) in the *left sidebar*.

In the *file and resource browser* (the left pane), you can find JumpStart options. From here you can choose to browse JumpStart for solutions, models, notebooks, and other resources, or you can view your currently launched solutions, endpoints, and training jobs.

To see what JumpStart has to offer, choose the JumpStart icon, and then choose **Browse JumpStart**. JumpStart opens in a new tab in the *main work area*. Here you can browse 1-click solutions, models, example notebooks, blogs, and video tutorials.



**Important**

Amazon SageMaker JumpStart makes certain content available from third-party sources. This content may be subject to separate license terms. You are responsible for reviewing and complying with any applicable license terms and making sure they are acceptable for your use case before downloading or using the content.

## Using JumpStart

At the top of the JumpStart page you can use search to look for topics of interest.

The screenshot shows the SageMaker JumpStart interface. At the top, the title "SageMaker JumpStart: Discover, train, and One-click solutions, pretrained models, and example notebooks" is displayed. Below the title, there is a search bar containing the text "pytorch". A list of search results is shown, each with a small icon and a title:

- ResNet 18
- SSD
- ResNet 34
- Organize, Track and Compare ML Trainings
- Training and Hosting a PyTorch Model
- Privacy for Sentiment Classification
- Maximizing NLP Model Performance
- ResNet 50

You can find JumpStart resources by using search, or by browsing each category that follows the search panel:

- **Solutions** – Launch end-to-end machine learning solutions that tie SageMaker to other Amazon services with one click.
- **Text models** – Deploy and fine-tune pretrained transformers for various natural language processing use cases.
- **Vision models** – Deploy and fine-tune pretrained models for image classification and object detection with one click.

- **SageMaker algorithms** – Train and deploy SageMaker built-in Algorithms for various problem types with these example notebooks.
- **Example notebooks** – Run example notebooks that use SageMaker features like spot instance training and experiments over a large variety of model types and use cases.
- **Blogs** – Read deep dives and solutions from machine learning experts hosted by Amazon.
- **Video tutorials** – Watch video tutorials for SageMaker features and machine learning use cases from machine learning experts hosted by Amazon.

## Solutions

When you choose a solution, JumpStart shows a description of the solution and a **Launch** button. When you click **Launch**, JumpStart creates all of the resources necessary to run the solution, including training and model hosting instances. After JumpStart launches the solution, JumpStart shows an **Open Notebook** button. You can click the button to use the provided notebooks and explore the solution's features. As artifacts are generated during launch or after running the provided notebooks, they are



listed in the **Generated Artifacts** table. You can delete individual Artifacts with the Trash icon ( ). You can delete all of the solution's resources by choosing **Delete solution resources**.

## Models

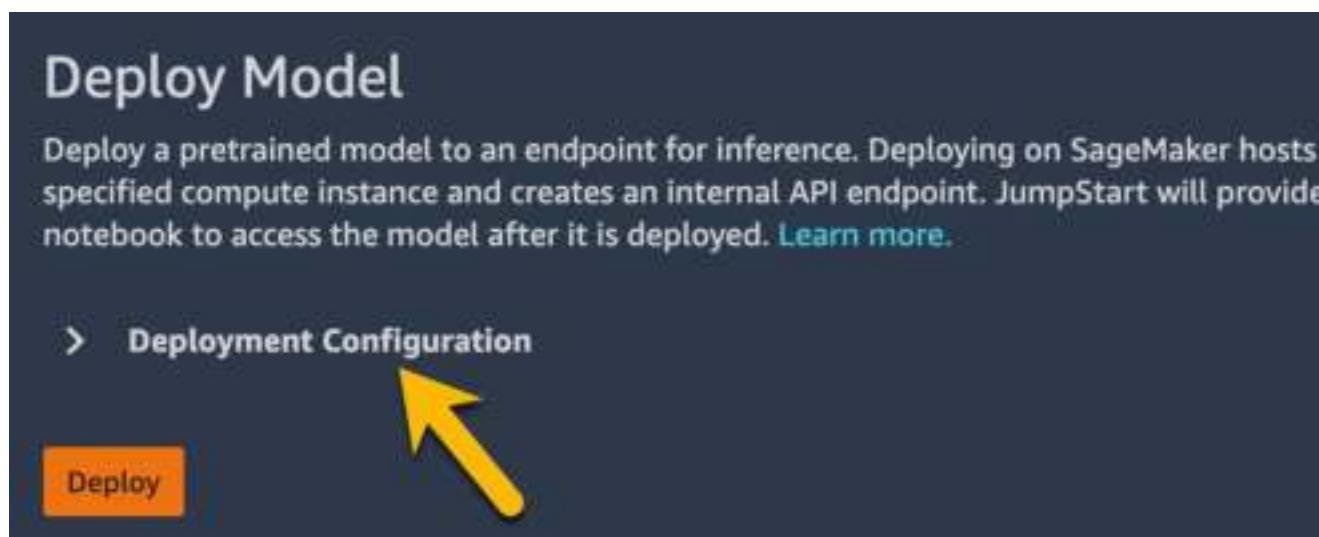
Models are available for quick deployment directly from JumpStart. You can also fine-tune some of these models. When you browse the models, you can scroll to the deploy and fine-tune sections to the **Description** section. In the **Description** section, you can learn more about the model, including what it can do with the model, what kind of inputs and outputs are expected, and the kind of data you need if you want to use transfer learning to fine-tune the model.

## Deploy a model

When you deploy a model from JumpStart, SageMaker hosts the model and deploys an endpoint that you can use for inference. JumpStart also provides an example notebook that you can use to access the model after it's deployed.

## Model Deployment Configuration

After you choose a model, the **Deploy Model** pane opens. Choose **Deployment Configuration** to configure your model deployment.



The default **Machine Type** for deploying a model depends on the model. The machine type is the hardware that the training job runs on. In the following example, the `m1.m5.large` instance is the default for this particular BERT model.

You can also change the **Endpoint Name**.

## Deploy Model

Deploy a pretrained model to an endpoint for inference. Deploying on SageMaker hosts a specified compute instance and creates an internal API endpoint. JumpStart will provide a notebook to access the model after it is deployed. [Learn more.](#)

### Deployment Configuration

Customize the machine type and endpoint name. [Learn more.](#)

#### Machine Type

ML.M5.Large

#### Endpoint Name

tf-tc-bert-en-uncased-l-12-h-768-a-12-2

[Reset to default](#)

[Deploy](#)

## Fine-Tune a Model

Fine-tuning trains a pretrained model on a new dataset without training from scratch. This process, also known as transfer learning, can produce accurate models with smaller datasets and less training time.

## Fine-Tuning Data Source

When you fine-tune a model, you can use the default dataset or choose your own data, which is located in an S3 bucket.



Top-5 model predictions: Labrador retriever, g  
Chesapeake Bay retriever, Weimaraner, Walker

## Fine-tune the Model on a New Dataset

The model can be fine-tuned to any given dataset comprising images belonging to any classes.

The model available for fine-tuning attaches a classification layer to the corresponding model available on TensorFlow, and initializes the layer parameters to random values. The dimension of the classification layer is determined based on the number of classes in the fine-tuning step fine-tunes the classification layer parameters while keeping the parameter feature extractor model frozen, and returns the fine-tuned model. The objective is to minimize error on the input data. The model returned by fine-tuning can be further deployed for inference. The following sections provide instructions for fine-tuning the model. The first section provides the instructions for how the training data should be formatted for input to the model.

- **Input:** A directory with as many sub-directories as the number of classes.
  - Each sub-directory should have images belonging to that class in .jpg format.
- **Output:** A trained model that can be deployed for inference.
  - A label mapping file is saved along with the trained model file on the s3 bucket.

We provide `tf_flowers` dataset as a default dataset for fine-tuning the model. `tf_flowers` dataset contains images of five types of flowers. The dataset has been downloaded from [TensorFlow](#). Apache License 2.0. Citation: @ONLINE {tfflowers, author = "The TensorFlow Team", title = "Flowers", month = "2019", url = "[http://download.tensorflow.org/example\\_images/flower\\_photos.tgz](http://download.tensorflow.org/example_images/flower_photos.tgz)"}

To browse the buckets available to you, choose **Find S3 bucket**. These buckets are limited by the permissions used to set up your Studio account. You can also specify an S3 URI by choosing **Enter S3 bucket location**.

## Fine-tune Model

Create a training job to fine-tune this pretrained model to fit your own data. Fine-tuning the model on a new dataset without training from scratch. It can produce accurate models with less training time. [Learn more](#).

▼ **Data Source**

Select the default dataset, or use your own data to fine-tune this model.

Default dataset     Find S3 bucket     Enter S3 bucket location

This option will fit the model to the default dataset. [Learn more](#).

*Default dataset:*

➤ Deployment Configuration

➤ Hyper-parameters

**Train**

**Tip**

To find out how to format the data in your bucket, choose [Learn more](#). Also, the description section for the model has detailed information about inputs and outputs.

For text models:

- The bucket must have a data.csv file.
- The first column must be a unique integer for the class label. For example: 1, 2, 3, 4, n.
- The second column must be a string.
- The second column should have the corresponding text that matches the type and language for the model.

For vision models:

- The bucket must have as many subdirectories as the number of classes.
- Each subdirectory should contain images that belong to that class in .jpg format.

**Note**

The S3 bucket must be in the same Amazon Region where you're running SageMaker Studio because SageMaker doesn't allow cross-region requests.

## Fine-Tuning deployment configuration

The p3 family is recommended as the fastest for deep learning training, and this is recommended for fine-tuning a model. The following chart shows the number of GPUs in each instance type. There are other available options that you can choose from, including p2 and g4 instance types.

Instance type	GPUs
p3.2xlarge	1
p3.8xlarge	4
p3.16xlarge	8
p3dn.24xlarge	8

## Hyperparameters

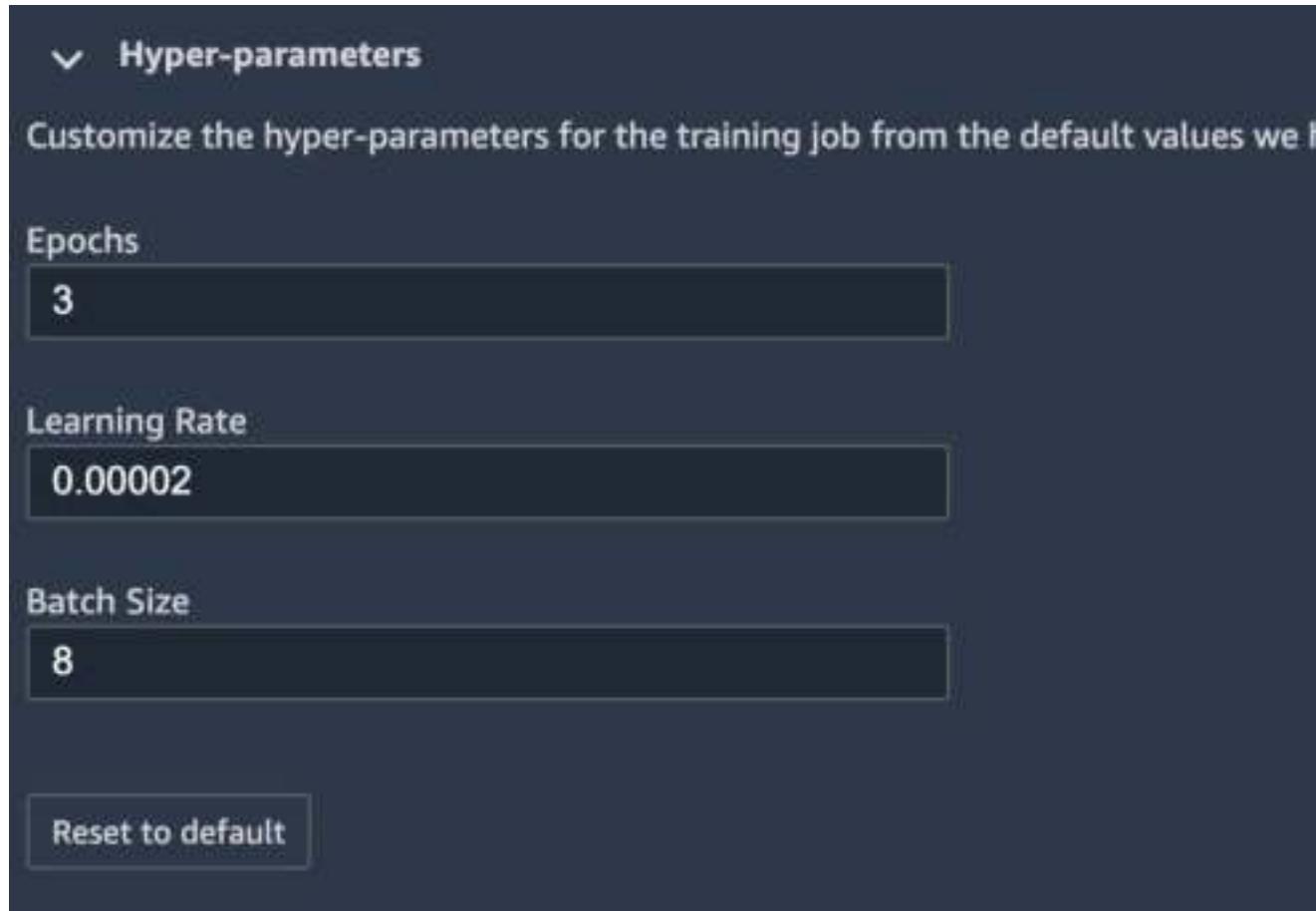
You can customize the hyperparameters of the training job that are used to fine-tune the model.

If you use the default dataset for text models without changing the hyperparameters, you get a nearly identical model as a result. For vision models, the default dataset is different from the dataset used to train the pretrained models, so your model is different as a result.

You have the following hyperparameter options:

- **Epochs** – One epoch is one cycle through the entire dataset. Multiple intervals complete a batch, and multiple batches eventually complete an epoch. Multiple epochs are run until the accuracy of the model reaches an acceptable level, or in other words, when the error rate drops below an acceptable level.
- **Learning rate** – The amount that values should be changed between epochs. As the model is refined, its internal weights are being nudged and error rates are checked to see if the model improves. A typical learning rate is 0.1 or 0.01, where 0.01 is a much smaller adjustment and could cause the training to take a long time to converge, whereas 0.1 is much larger and can cause the training to overshoot. It is one of the primary hyperparameters that you might adjust for training your model. Note that for text models, a much smaller learning rate (5e-5 for BERT) can result in a more accurate model.
- **Batch size** – The number of records from the dataset that to be selected for each interval to send to the GPUs available in training. In an image example, you might send out 32 images per GPU, so 32 would be your batch size. If you choose an instance type with more than one GPU, the batch is divided by the number of GPUs. Suggested batch size varies depending on the data and the model that you are using. For example, how you optimize for image data differs from how you handle language data. In the instance type chart in the deployment configuration section, you can see the number of GPUs per instance type. Start with a standard recommended batch size (for example, 32 for a vision model). Then, multiply this by the number of GPUs in the instance type that you selected. For example, if

you're using a p3.8xlarge, this would be  $32(\text{batch size}) * 4(\text{GPUs})$ , for a total of 128 as your batch size adjusted for the number of GPUs. For a text model like BERT, try starting with a batch size of 64, and then reduce as needed.



## Training Output

When the fine-tuning process is complete, JumpStart provides information about the model: parent model, training job name, training job Amazon Resource Name (ARN), training time, and output path. The output path is where you can find your new model in an S3 bucket. The folder structure uses the model name you provided and the model file is in an /output subfolder and it's always named `model.tar.gz`.

Example: `s3://bucket/model-name/output/model.tar.gz`

## Next Steps

For a deep dive into Studio features:

- [Amazon SageMaker Studio Tour \(p. 51\)](#) – An end-to-end tour of the main features of SageMaker Studio.

## Amazon SageMaker Studio Tour

For a walkthrough that takes you on a tour of the main features of Amazon SageMaker Studio, see the [xgboost\\_customer\\_churn\\_studio.ipynb](#) sample notebook from the [aws/amazon-sagemaker-examples](#) repository. The code in the notebook trains multiple models and sets up the SageMaker Debugger and SageMaker Model Monitor. The walkthrough shows you how to view the trials, compare the resulting models, show the debugger results, and deploy the best model using the SageMaker Studio UI. You don't need to understand the code to follow this walkthrough.

For a series of videos that shows how to use the main features of SageMaker Studio, see [NEW! Amazon SageMaker Studio](#) on YouTube.

### Prerequisites

To run the notebook for this tour, you need:

- An Amazon Web Services SSO or IAM account to sign in to Studio. For information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
- Basic familiarity with the Studio user interface and Jupyter notebooks. For information, see [Amazon SageMaker Studio UI Overview \(p. 69\)](#).
- A copy of the [aws/amazon-sagemaker-examples](#) repository in your Studio environment.

### To clone the repository

1. Sign in to SageMaker Studio. For Amazon Web Services SSO users, sign in using the URL from your invitation email. For IAM users, follow these steps.
  - a. Sign in to the [SageMaker console](#).
  - b. Choose **Amazon SageMaker Studio** in the left navigation pane.
  - c. Choose **Open Studio** in the row next to your user name.
2. On the top menu, choose **File** then **New** then **Terminal**.
3. At the command prompt, run the following command.

```
git clone https://github.com/aws/amazon-sagemaker-examples.git
```

### Note

If you encounter an error when you run the sample notebook, and some time has passed from when you cloned the repository, review the notebook on the remote repository for updates.

## Get Started with Amazon SageMaker Notebook Instances

One of the best ways for machine learning (ML) practitioners to use Amazon SageMaker is to train and deploy ML models using SageMaker notebook instances. The SageMaker notebook instances help create the environment by initiating Jupyter servers on Amazon Elastic Compute Cloud (Amazon EC2) and providing preconfigured kernels with the following packages: the Amazon SageMaker Python SDK, Amazon SDK for Python (Boto3), Amazon Command Line Interface (Amazon CLI), Conda, Pandas, deep learning framework libraries, and other libraries for data science and machine learning.

# Machine Learning with the SageMaker Python SDK

To train, validate, deploy, and evaluate an ML model in a SageMaker notebook instance, use the SageMaker Python SDK. The SageMaker Python SDK abstracts Amazon SDK for Python (Boto3) and SageMaker API operations. It enables you to integrate with and orchestrate other Amazon services, such as Amazon Simple Storage Service (Amazon S3) for saving data and model artifacts, Amazon Elastic Container Registry (ECR) for importing and servicing the ML models, Amazon Elastic Compute Cloud (Amazon EC2) for training and inference.

You can also take advantage of SageMaker features that help you deal with every stage of a complete ML cycle: data labeling, data preprocessing, model training, model deployment, evaluation on prediction performance, and monitoring the quality of model in production.

If you're a first-time SageMaker user, we recommend you to use the SageMaker Python SDK, following the end-to-end ML tutorial. To find the open source documentation, see the [Amazon SageMaker Python SDK](#).

## Tutorial Overview

This Get Started tutorial walks you through how to create a SageMaker notebook instance, open a Jupyter notebook with a preconfigured kernel with the Conda environment for machine learning, and start a SageMaker session to run an end-to-end ML cycle. You'll learn how to save a dataset to a default Amazon S3 bucket automatically paired with the SageMaker session, submit a training job of an ML model to Amazon EC2, and deploy the trained model for prediction by hosting or batch inferencing through Amazon EC2.

This tutorial explicitly shows a complete ML flow of training the XGBoost model from the SageMaker built-in model pool. You use the [US Adult Census dataset](#), and you evaluate the performance of the trained SageMaker XGBoost model on predicting individuals' income.

- [SageMaker XGBoost](#) – The [XGBoost](#) model is adapted to the SageMaker environment and preconfigured as Docker containers. SageMaker provides a suite of [built-in algorithms](#) that are prepared for using SageMaker features. To learn more about what ML algorithms are adapted to SageMaker, see [Choose an Algorithm](#) and [Use Amazon SageMaker Built-in Algorithms](#). For the SageMaker built-in algorithm API operations, see [First-Party Algorithms](#) in the [Amazon SageMaker Python SDK](#).
- [Adult Census dataset](#) – The dataset from the [1994 Census bureau database](#) by Ronny Kohavi and Barry Becker (Data Mining and Visualization, Silicon Graphics). The SageMaker XGBoost model is trained using this dataset to predict if an individual makes over \$50,000 a year or less.

### Topics

- [Step 1: Create an Amazon SageMaker Notebook Instance \(p. 53\)](#)
- [Step 2: Create a Jupyter Notebook \(p. 54\)](#)
- [Step 3: Download, Explore, and Transform a Dataset \(p. 55\)](#)
- [Step 4: Train a Model \(p. 59\)](#)
- [Step 5: Deploy the Model to Amazon EC2 \(p. 63\)](#)
- [Step 6: Evaluate the Model \(p. 65\)](#)
- [Step 7: Clean Up \(p. 68\)](#)

# Step 1: Create an Amazon SageMaker Notebook Instance

An Amazon SageMaker notebook instance is a fully managed machine learning (ML) Amazon Elastic Compute Cloud (Amazon EC2) compute instance that runs the Jupyter Notebook App. You use the notebook instance to create and manage Jupyter notebooks for preprocessing data and to train and deploy machine learning models.

## To create a SageMaker notebook instance

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Choose **Notebook instances**, and then choose **Create notebook instance**.
3. On the **Create notebook instance** page, provide the following information (if a field is not mentioned, leave the default values):
  - a. For **Notebook instance name**, type a name for your notebook instance.
  - b. For **Instance type**, choose `m1.t2.medium`. This is the least expensive instance type that notebook instances support, and it suffices for this exercise. If a `m1.t2.medium` instance type isn't available in your current Amazon Region, choose `m1.t3.medium`.
  - c. For **IAM role**, choose **Create a new role**, and then choose **Create role**. This IAM role automatically gets permissions to access any S3 bucket that has `sagemaker` in the name. It gets these permissions through the `AmazonSageMakerFullAccess` policy, which SageMaker attaches to the role.

### Note

If you want to grant the IAM role permission to access S3 buckets without `sagemaker` in the name, you need to attach the `S3FullAccess` policy or limit the permissions to specific S3 buckets to the IAM role. For more information and examples of adding bucket policies to the IAM role, see [Bucket Policy Examples](#).

- d. Choose **Create notebook instance**.

In a few minutes, SageMaker launches an ML compute instance—in this case, a notebook instance—and attaches a 5 GB of Amazon EBS storage volume to it. The notebook instance has a preconfigured Jupyter notebook server, SageMaker and Amazon SDK libraries, and a set of Anaconda libraries.

For more information about creating a SageMaker notebook instance, see [Create a Notebook Instance](#).

## (Optional) Change SageMaker Notebook Instance Settings

If you want to change the ML compute instance type or the size of the Amazon EBS storage of a SageMaker notebook instance that's already created, you can edit the notebook instance settings.

### To change and update the SageMaker Notebook instance type and the EBS volume

1. On the **Notebook instances** page in the SageMaker console, choose your notebook instance.
2. Choose **Actions**, choose **Stop**, and then wait until the notebook instance fully stops.
3. After the notebook instance status changes to **Stopped**, choose **Actions**, and then choose **Update setting**.
  - a. For **Notebook instance type**, choose a different ML instance type.
  - b. For **Volume size in GB**, type a different integer to specify a new EBS volume size.

#### Note

EBS storage volumes are encrypted, so SageMaker can't determine the amount of available free space on the volume. Because of this, you can increase the volume size when you update a notebook instance, but you can't decrease the volume size. If you want to decrease the size of the ML storage volume in use, create a new notebook instance with the desired size.

4. At the bottom of the page, choose **Update notebook instance**.
5. When the update is complete, **Start** the notebook instance with the new settings.

For more information about updating SageMaker notebook instance settings, see [Update a Notebook Instance](#).

## (Optional) Advanced Settings for SageMaker Notebook Instances

The following tutorial video shows how to set up and use SageMaker notebook instances through the SageMaker console with advanced options, such as SageMaker lifecycle configuration and importing GitHub repositories. (Length: 26:04)

For complete documentation about SageMaker notebook instance, see [Use Amazon SageMaker notebook Instances](#).

## Step 2: Create a Jupyter Notebook

To start scripting for training and deploying your model, create a Jupyter notebook in the SageMaker notebook instance. Using the Jupyter notebook, you can conduct machine learning (ML) experiments for training and inference while accessing the SageMaker features and the Amazon infrastructure.

### To create a Jupyter notebook

1. Open the notebook instance as follows:
  - a. Sign in to the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
  - b. On the **Notebook instances** page, open your notebook instance by choosing either **Open JupyterLab** for the JupyterLab interface or **Open Jupyter** for the classic Jupyter view.

#### Note

If the notebook instance status shows **Pending** in the **Status** column, your notebook instance is still being created. The status will change to **InService** when the notebook instance is ready for use.

2. Create a notebook as follows:
  - If you opened the notebook in the JupyterLab view, on the **File** menu, choose **New**, and then choose **Notebook**. For **Select Kernel**, choose **conda\_python3**. This preinstalled environment includes the default Anaconda installation and Python 3.
  - If you opened the notebook in the classic Jupyter view, on the **Files** tab, choose **New**, and then choose **conda\_python3**. This preinstalled environment includes the default Anaconda installation and Python 3.
3. Save the notebooks as follows:
  - In the JupyterLab view, choose **File**, choose **Save Notebook As...**, and then rename the notebook.
  - In the Jupyter classic view, choose **File**, choose **Save as...**, and then rename the notebook.

## Step 3: Download, Explore, and Transform a Dataset

In this step, you load the [Adult Census dataset](#) to your notebook instance using the [SHAP \(SHapley Additive exPlanations\) Library](#), review the dataset, transform it, and upload it to Amazon S3.

To run the following example, paste the sample code into a cell in your notebook instance.

### Load Adult Census Dataset Using SHAP

Using the SHAP library, import the Adult Census dataset as shown following:

```
import shap
X, y = shap.datasets.adult()
X_display, y_display = shap.datasets.adult(display=True)
feature_names = list(X.columns)
feature_names
```

#### Note

If the current Jupyter kernel does not have the SHAP library, install it by running the following conda command:

```
%conda install -c conda-forge shap
```

If you're using JupyterLab, you must manually refresh the kernel after the installation and updates have completed. Run the following IPython script to shut down the kernel (the kernel will restart automatically):

```
import IPython
IPython.Application.instance().kernel.do_shutdown(True)
```

The `feature_names` list object should return the following list of features:

```
['Age',
 'Workclass',
 'Education-Num',
 'Marital Status',
 'Occupation',
 'Relationship',
 'Race',
 'Sex',
 'Capital Gain',
 'Capital Loss',
 'Hours per week',
 'Country']
```

#### Tip

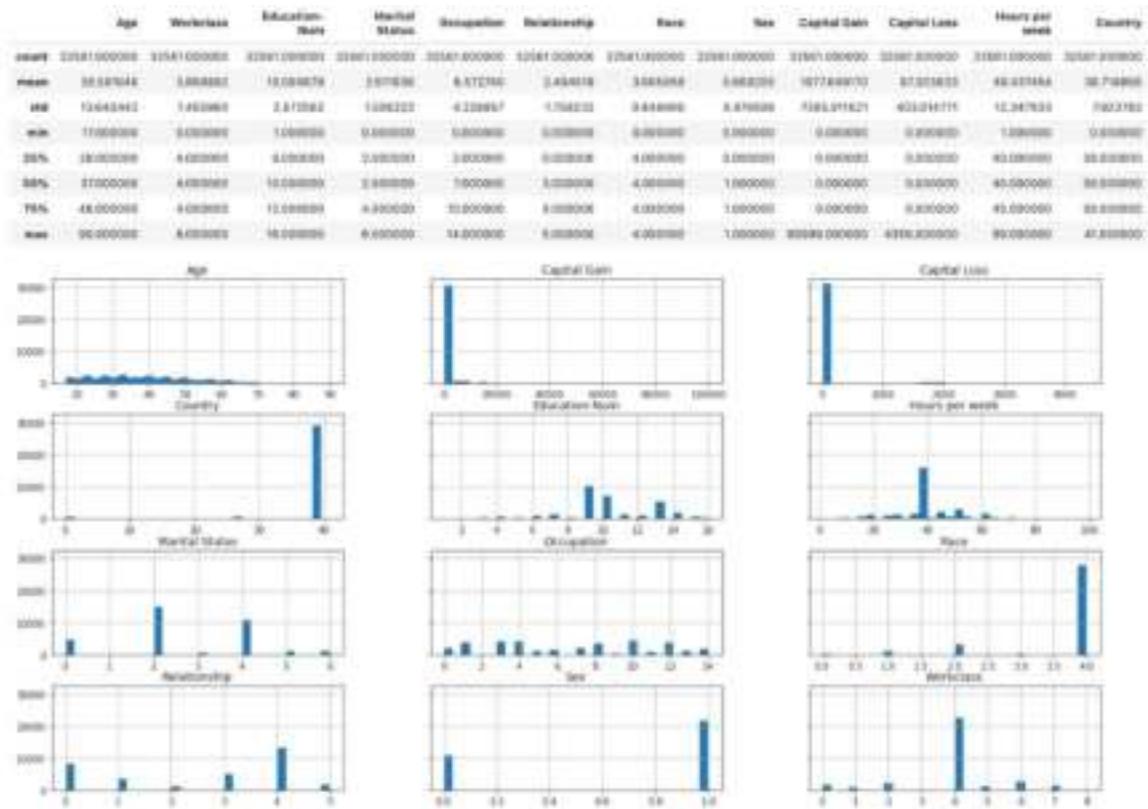
If you're starting with unlabeled data, you can use Amazon SageMaker Ground Truth to create a data labeling workflow in minutes. To learn more, see [Label Data](#).

### Overview the Dataset

Run the following script to display the statistical overview of the dataset and histograms of the numeric features.

```
display(X.describe())
```

```
hist = X.hist(bins=30, sharey=True, figsize=(20, 10))
```



### Tip

If you want to use a dataset that needs to be cleaned and transformed, you can simplify and streamline data preprocessing and feature engineering using Amazon SageMaker Data Wrangler. To learn more, see [Prepare ML Data with Amazon SageMaker Data Wrangler](#).

## Split the Dataset into Train, Validation, and Test Datasets

Using Sklearn, split the dataset into a training set and a test set. The dataset is randomly sorted with the fixed random seed: 80 percent of the dataset for training set and 20 percent of it for a test set.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
X_train_display = X_display.loc[X_train.index]
```

Split the training set to separate out a validation set. 75 percent of the training set becomes the final training set, and the rest is the validation set.

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25,
random_state=1)
X_train_display = X_display.loc[X_train.index]
X_val_display = X_display.loc[X_val.index]
```

Using the pandas package, explicitly align each dataset by concatenating the numeric features with the true labels.

```
import pandas as pd
```

```

train = pd.concat([pd.Series(y_train, index=X_train.index,
                             name='Income>50K', dtype=int), X_train], axis=1)
validation = pd.concat([pd.Series(y_val, index=X_val.index,
                                  name='Income>50K', dtype=int), X_val], axis=1)
test = pd.concat([pd.Series(y_test, index=X_test.index,
                           name='Income>50K', dtype=int), X_test], axis=1)

```

Check if the dataset is split and structured as expected:

**train**

Income>50K	Age	Workclass	Education-Num	Marital Status	Occupation	Relationship	Race	Sex	Capital Gain	Capital Loss	Hours per week	Country
10911	1	47.0	4	9.0	2	3	4	4	1	0.0	0.0	40.0
17852	0	31.0	4	13.0	2	3	4	3	1	0.0	0.0	38.0
29566	1	32.0	4	9.0	2	13	3	4	0	0.0	0.0	32.0
30287	0	68.0	4	9.0	2	3	4	2	1	0.0	0.0	40.0
34019	0	17.0	4	8.0	4	0	3	4	1	0.0	0.0	33.0
—	—	—	—	—	—	—	—	—	—	—	—	—
21168	0	43.0	4	8.0	2	14	4	4	1	0.0	0.0	40.0
6452	0	26.0	4	9.0	4	7	0	4	1	0.0	0.0	31.0
31352	0	32.0	7	14.0	2	10	4	4	1	0.0	0.0	30.0
6678	0	45.0	4	9.0	4	6	0	4	1	0.0	0.0	40.0
23804	0	23.0	4	8.0	4	1	1	4	0	0.0	0.0	40.0

19536 rows × 13 columns

**validation**

Income>50K	Age	Workclass	Education-Num	Marital Status	Occupation	Relationship	Race	Sex	Capital Gain	Capital Loss	Hours per week	Country
16530	0	25.0	4	4.0	2	0	4	4	1	0.0	0.0	40.0
26723	0	41.0	6	9.0	2	6	0	4	0	0.0	0.0	40.0
3338	0	78.0	0	9.0	6	0	0	2	0	0.0	0.0	30.0
19367	1	43.0	2	15.0	2	10	4	4	1	15024.0	0.0	40.0
30274	0	31.0	5	8.0	4	12	2	4	1	0.0	0.0	40.0
—	—	—	—	—	—	—	—	—	—	—	—	—
1604	0	46.0	7	9.0	2	13	4	4	1	0.0	0.0	40.0
8837	1	71.0	4	10.0	6	12	0	4	1	0.0	0.0	30.0
11034	0	36.0	4	9.0	5	14	2	4	1	0.0	0.0	40.0
2819	0	31.0	4	9.0	4	8	0	4	0	0.0	0.0	40.0
64152	1	37.0	4	10.0	2	12	4	4	1	0.0	0.0	30.0

6512 rows × 13 columns

**test**

Income>50K	Age	Workclass	Education-Num	Marital Status	Occupation	Relationship	Race	Sex	Capital Gain	Capital Loss	Hours per week	Country
0646	0	32.0	0	4.0	0	0	0	0	0.0	0.0	66.0	29
708	0	16.0	4	3.0	4	0	0	1	0.0	0.0	25.0	39
7385	1	25.0	4	13.0	4	0	0	1	27629.0	0.0	50.0	39
18671	0	33.0	4	9.0	2	10	0	1	0.0	0.0	41.0	39
21932	0	36.0	4	7.0	0	7	1	0	0.0	0.0	40.0	39
...	...	...	...	...	...	...	...	...	...	...	...	...
6829	1	39.0	4	13.0	2	10	0	1	0.0	0.0	20.0	39
26723	0	17.0	4	6.0	4	12	0	0	0.0	0.0	20.0	39
29614	0	36.0	4	9.0	4	14	0	1	0.0	0.0	40.0	39
1600	0	30.0	4	7.0	2	3	0	1	0.0	0.0	40.0	39
639	1	32.0	0	16.0	2	10	0	1	0.0	0.0	60.0	39

6513 rows × 13 columns

## Convert the Train and Validation Datasets to CSV Files

Convert the `train` and `validation` dataframe objects to CSV files to match the input file format for the XGBoost algorithm.

```
# Use 'csv' format to store the data
# The first column is expected to be the output column
train.to_csv('train.csv', index=False, header=False)
validation.to_csv('validation.csv', index=False, header=False)
```

## Upload the Datasets to Amazon S3

Using the SageMaker and Boto3, upload the training and validation datasets to the default Amazon S3 bucket. The datasets in the S3 bucket will be used by a compute-optimized SageMaker instance on Amazon EC2 for training.

The following code sets up the default S3 bucket URL for your current SageMaker session, creates a new `demo-sagemaker-xgboost-adult-income-prediction` folder, and uploads the training and validation datasets to the data subfolder.

```
import sagemaker, boto3, os
bucket = sagemaker.Session().default_bucket()
prefix = "demo-sagemaker-xgboost-adult-income-prediction"

boto3.Session().resource('s3').Bucket(bucket).Object(
    os.path.join(prefix, 'data/train.csv')).upload_file('train.csv')
boto3.Session().resource('s3').Bucket(bucket).Object(
    os.path.join(prefix, 'data/validation.csv')).upload_file('validation.csv')
```

Run the following Amazon CLI to check if the CSV files are successfully uploaded to the S3 bucket.

```
! aws s3 ls {bucket}/{prefix}/data --recursive
```

This should return the following output:

```
2021-01-14 17:52:09      786285 demo-sagemaker-xgboost-adult-income-prediction/data/train.csv
2021-01-14 17:52:10      262122 demo-sagemaker-xgboost-adult-income-prediction/data/validation.csv
```

## Step 4: Train a Model

The [Amazon SageMaker Python SDK](#) provides framework estimators and generic estimators to train your model while orchestrating the machine learning (ML) lifecycle accessing the SageMaker features for training and the Amazon infrastructures, such as Amazon Elastic Container Registry (Amazon ECR), Amazon Elastic Compute Cloud (Amazon EC2), Amazon Simple Storage Service (Amazon S3). For more information about SageMaker built-in framework estimators and algorithm estimators, see [Frameworks](#) and [First-Party Algorithms](#) respectively in the [Amazon SageMaker Python SDK](#) documentation.

### Topics

- [Choose the Training Algorithm \(p. 59\)](#)
- [Create and Run a Training Job \(p. 59\)](#)

## Choose the Training Algorithm

To choose the right algorithm for your dataset, you typically need to evaluate different models to find the most suitable models to your data. For simplicity, the SageMaker [XGBoost Algorithm \(p. 1524\)](#) built-in algorithm is used throughout this tutorial without the pre-evaluation of models.

### Tip

If you want SageMaker to find an appropriate model for your tabular dataset, use Amazon SageMaker Autopilot that automates a machine learning solution. For more information, see [Automate model development with Amazon SageMaker Autopilot \(p. 142\)](#).

## Create and Run a Training Job

After you figured out which model to use, start constructing a SageMaker estimator for training. This tutorial uses the XGBoost built-in algorithm for the SageMaker generic estimator.

### To run a model training job

1. Import the [Amazon SageMaker Python SDK](#) and start by retrieving the basic information from your current SageMaker session.

```
import sagemaker

region = sagemaker.Session().boto_region_name
print("Amazon Region: {}".format(region))

role = sagemaker.get_execution_role()
print("RoleArn: {}".format(role))
```

This returns the following information:

- **region** – The current Amazon Region where the SageMaker notebook instance is running.
- **role** – The IAM role used by the notebook instance.

### Note

Check the SageMaker Python SDK version by running `sagemaker.__version__`. This tutorial is based on `sagemaker>=2.20`. If the SDK is outdated, install the latest version by running the following command:

```
! pip install -qU sagemaker
```

If you run this installation in your existing SageMaker Studio or notebook instances, you need to manually refresh the kernel to finish applying the version update.

2. Create an XGBoost estimator using the `sagemaker.estimator.Estimator` class. In the following example code, the XGBoost estimator is named `xgb_model`.

```
from sagemaker.debugger import Rule, rule_configs
from sagemaker.session import TrainingInput

s3_output_location='s3://{}//{}//{}'.format(bucket, prefix, 'xgboost_model')

container=sagemaker.image_uris.retrieve("xgboost", region, "1.2-1")
print(container)

xgb_model=sagemaker.estimator.Estimator(
    image_uri=container,
    role=role,
    instance_count=1,
    instance_type='ml.m4.xlarge',
    volume_size=5,
    output_path=s3_output_location,
    sagemaker_session=sagemaker.Session(),
    rules=[Rule.sagemaker(rule_configs.create_xgboost_report())]
)
```

To construct the SageMaker estimator, specify the following parameters:

- `image_uri` – Specify the training container image URI. In this example, the SageMaker XGBoost training container URI is specified using `sagemaker.image_uris.retrieve`.
- `role` – The Amazon Identity and Access Management (IAM) role that SageMaker uses to perform tasks on your behalf (for example, reading training results, call model artifacts from Amazon S3, and writing training results to Amazon S3).
- `instance_count` and `instance_type` – The type and number of Amazon EC2 ML compute instances to use for model training. For this training exercise, you use a single `ml.m4.xlarge` instance, which has 4 CPUs, 16 GB of memory, an Amazon Elastic Block Store (Amazon EBS) storage, and a high network performance. For more information about EC2 compute instance types, see [Amazon EC2 Instance Types](#). For more information about billing, see [Amazon SageMaker pricing](#).
- `train_volume_size` – The size, in GB, of the EBS storage volume to attach to the training instance. This must be large enough to store training data if you use `File mode` (`File mode` is on by default).
- `output_path` – The path to the S3 bucket where SageMaker stores the model artifact and training results.
- `sagemaker_session` – The session object that manages interactions with SageMaker API operations and other Amazon service that the training job uses.
- `rules` – Specify a list of SageMaker Debugger built-in rules. In this example, the `create_xgboost_report()` rule creates an XGBoost report that provides insights into the training progress and results. For more information, see [SageMaker Debugger XGBoost Training Report \(p. 1711\)](#).

### Tip

If you want to run distributed training of large sized deep learning models, such as convolutional neural networks (CNN) and natural language processing (NLP) models, use SageMaker Distributed for data parallelism or model parallelism. For more information, see [Distributed Training \(p. 1771\)](#).

3. Set the hyperparameters for the XGBoost algorithm by calling the `set_hyperparameters` method of the estimator. For a complete list of XGBoost hyperparameters, see [XGBoost Hyperparameters \(p. 1531\)](#).

```
xgb_model.set_hyperparameters(  
    max_depth = 5,  
    eta = 0.2,  
    gamma = 4,  
    min_child_weight = 6,  
    subsample = 0.7,  
    objective = "binary:logistic",  
    num_round = 1000  
)
```

**Tip**

You can also tune the hyperparameters using the SageMaker hyperparameter optimization feature. For more information, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

4. Use the `TrainingInput` class to configure a data input flow for training. The following example code shows how to configure `TrainingInput` objects to use the training and validation datasets you uploaded to Amazon S3 in the [Split the Dataset into Train, Validation, and Test Datasets \(p. 56\)](#) section.

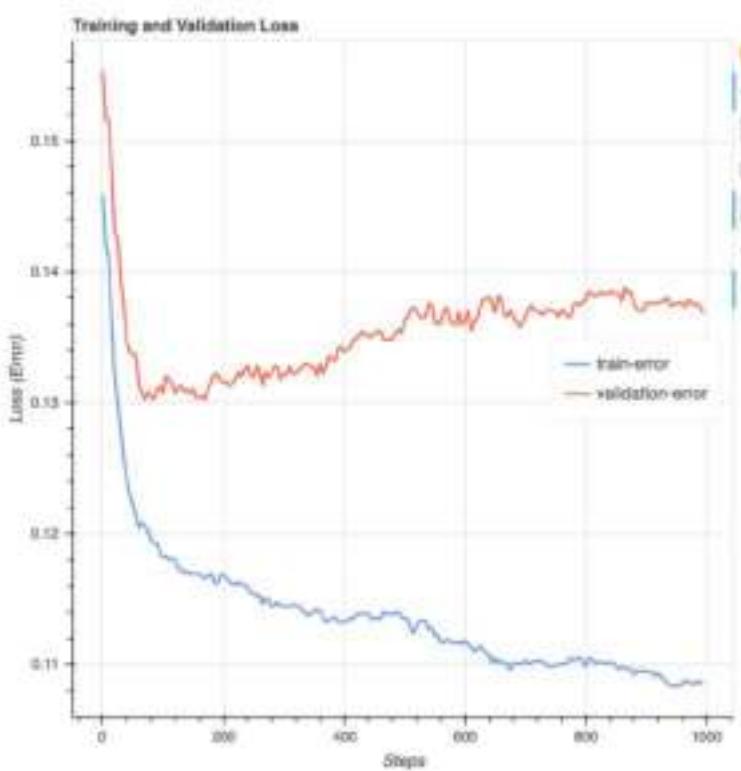
```
from sagemaker.session import TrainingInput  
  
train_input = TrainingInput(  
    "s3://{}//{}//{}".format(bucket, prefix, "data/train.csv"), content_type="csv"  
)  
validation_input = TrainingInput(  
    "s3://{}//{}//{}".format(bucket, prefix, "data/validation.csv"), content_type="csv"  
)
```

5. To start model training, call the estimator's `fit` method with the training and validation datasets. By setting `wait=True`, the `fit` method displays progress logs and waits until training is complete.

```
xgb_model.fit({"train": train_input, "validation": validation_input}, wait=True)
```

For more information about model training, see [Train a Model with Amazon SageMaker \(p. 8\)](#). This tutorial training job might take up to 10 minutes.

After the training job has done, you can download an XGBoost training report and a profiling report generated by SageMaker Debugger. The XGBoost training report offers you insights into the training progress and results, such as the loss function with respect to iteration, feature importance, confusion matrix, accuracy curves, and other statistical results of training. For example, you can find the following loss curve from the XGBoost training report which clearly indicates that there is an overfitting problem.



Run the following code to specify the S3 bucket URI where the Debugger training reports are generated and check if the reports exist.

```
rule_output_path = xgb_model.output_path + "/" + xgb_model.latest_training_job.name +  
    "/rule-output"  
! aws s3 ls {rule_output_path} --recursive
```

Download the Debugger XGBoost training and profiling reports to the current workspace:

```
! aws s3 cp {rule_output_path} ./ --recursive
```

Run the following IPython script to get the file link of the XGBoost training report:

```
from IPython.display import FileLink, FileLinks  
display("Click link below to view the XGBoost Training report",  
    FileLink("CreateXgboostReport/xgboost_report.html"))
```

The following IPython script returns the file link of the Debugger profiling report that shows summaries and details of the EC2 instance resource utilization, system bottleneck detection results, and python operation profiling results:

```
profiler_report_name = [rule["RuleConfigurationName"]  
    for rule in xgb_model.latest_training_job.rule_job_summary()  
        if "Profiler" in rule["RuleConfigurationName"]][0]  
profiler_report_name  
display("Click link below to view the profiler report", FileLink(profiler_report_name  
    +"/profiler-output/profiler-report.html"))
```

**Tip**

If the HTML reports do not render plots in the JupyterLab view, you must choose **Trust HTML** at the top of the reports.

To identify training issues, such as overfitting, vanishing gradients, and other problems that prevents your model from converging, use SageMaker Debugger and take automated actions while prototyping and training your ML models. For more information, see [Amazon SageMaker Debugger \(p. 1579\)](#). To find a complete analysis of model parameters, see the [Explainability with Amazon SageMaker Debugger](#) example notebook.

You now have a trained XGBoost model. SageMaker stores the model artifact in your S3 bucket. To find the location of the model artifact, run the following code to print the `model_data` attribute of the `xgb_model` estimator:

```
xgb_model.model_data
```

**Tip**

To measure biases that can occur during each stage of the ML lifecycle (data collection, model training and tuning, and monitoring of ML models deployed for prediction), use SageMaker Clarify. For more information, see [Model Explainability \(p. 1852\)](#). For an end-to-end example of it, see the [Fairness and Explainability with SageMaker Clarify](#) example notebook.

## Step 5: Deploy the Model to Amazon EC2

To get predictions, deploy your model to Amazon EC2 using Amazon SageMaker.

**Topics**

- [Deploy the Model to SageMaker Hosting Services \(p. 63\)](#)
- [\(Optional\) Use SageMaker Predictor to Reuse the Hosted Endpoint \(p. 64\)](#)
- [\(Optional\) Make Prediction with Batch Transform \(p. 64\)](#)

## Deploy the Model to SageMaker Hosting Services

To host a model through Amazon EC2 using Amazon SageMaker, deploy the model that you trained in [Create and Run a Training Job \(p. 59\)](#) by calling the `deploy` method of the `xgb_model` estimator. When you call the `deploy` method, you must specify the number and type of EC2 ML instances that you want to use for hosting an endpoint.

```
import sagemaker
from sagemaker.serializers import CSVSerializer
xgb_predictor=xgb_model.deploy(
    initial_instance_count=1,
    instance_type='ml.t2.medium',
    serializer=CSVSerializer()
)
```

- `initial_instance_count` (int) – The number of instances to deploy the model.
- `instance_type` (str) – The type of instances that you want to operate your deployed model.
- `serializer` (int) – Serialize input data of various formats (a NumPy array, list, file, or buffer) to a CSV-formatted string. We use this because the XGBoost algorithm accepts input files in CSV format.

The `deploy` method creates a deployable model, configures the SageMaker hosting services endpoint, and launches the endpoint to host the model. For more information, see the [SageMaker generic](#)

Estimator's `deploy` class method in the [Amazon SageMaker Python SDK](#). To retrieve the name of endpoint that's generated by the `deploy` method, run the following code:

```
xgb_predictor.endpoint_name
```

This should return the endpoint name of the `xgb_predictor`. The format of the endpoint name is "`sagemaker-xgboost-YYYY-MM-DD-HH-MM-SS-SSS`". This endpoint stays active in the ML instance, and you can make instantaneous predictions at any time unless you shut it down later. Copy this endpoint name and save it to reuse and make real-time predictions elsewhere in SageMaker Studio or SageMaker notebook instances.

**Tip**

To learn more about compiling and optimizing your model for deployment to Amazon EC2 instances or edge devices, see [Compile and Deploy Models with Neo](#).

## (Optional) Use SageMaker Predictor to Reuse the Hosted Endpoint

After you deploy the model to an endpoint, you can set up a new SageMaker predictor by pairing the endpoint and continuously make real-time predictions in any other notebooks. The following example code shows how to use the SageMaker Predictor class to set up a new predictor object using the same endpoint. Re-use the endpoint name that you used for the `xgb_predictor`.

```
import sagemaker
xgb_predictor_reuse=sagemaker.predictor.Predictor(
    endpoint_name="sagemaker-xgboost-YYYY-MM-DD-HH-MM-SS-SSS",
    sagemaker_session=sagemaker.Session(),
    serializer=sagemaker.serializers.CSVSerializer()
)
```

The `xgb_predictor_reuse` Predictor behaves exactly the same as the original `xgb_predictor`. For more information, see the [SageMaker Predictor](#) class in the [Amazon SageMaker Python SDK](#).

## (Optional) Make Prediction with Batch Transform

Instead of hosting an endpoint in production, you can run a one-time batch inference job to make predictions on a test dataset using the SageMaker batch transform. After your model training has completed, you can extend the estimator to a `transformer` object, which is based on the [SageMaker Transformer](#) class. The batch transformer reads in input data from a specified S3 bucket and makes predictions.

### To run a batch transform job

- Run the following code to convert the feature columns of the test dataset to a CSV file and uploads to the S3 bucket:

```
X_test.to_csv('test.csv', index=False, header=False)
boto3.Session().resource('s3').Bucket(bucket).Object(
    os.path.join(prefix, 'test/test.csv')).upload_file('test.csv')
```

- Specify S3 bucket URLs of input and output for the batch transform job as shown following:

```
# The location of the test dataset
batch_input = 's3://{}//{}//test'.format(bucket, prefix)
```

```
# The location to store the results of the batch transform job
batch_output = 's3://{}{}/batch-prediction'.format(bucket, prefix)
```

3. Create a transformer object specifying the minimal number of parameters: the `instance_count` and `instance_type` parameters to run the batch transform job, and the `output_path` to save prediction data as shown following:

```
transformer = xgb_model.transformer(
    instance_count=1,
    instance_type='ml.m4.xlarge',
    output_path=batch_output
)
```

4. Initiate the batch transform job by executing the `transform()` method of the `transformer` object as shown following:

```
transformer.transform(
    data=batch_input,
    data_type='S3Prefix',
    content_type='text/csv',
    split_type='Line'
)
transformer.wait()
```

5. When the batch transform job is complete, SageMaker creates the `test.csv.out` prediction data saved in the `batch_output` path, which should be in the following format: `s3://sagemaker-<region>-111122223333/demo-sagemaker-xgboost-adult-income-prediction/batch-prediction`. Run the following Amazon CLI to download the output data of the batch transform job:

```
! aws s3 cp {batch_output} ./ --recursive
```

This should create the `test.csv.out` file under the current working directory. You'll be able to see the float values that are predicted based on the logistic regression of the XGBoost training job.

## Step 6: Evaluate the Model

Now that you have trained and deployed a model using Amazon SageMaker, evaluate the model to ensure that it generates accurate predictions on new data. For model evaluation, use the test dataset that you created in [Step 3: Download, Explore, and Transform a Dataset \(p. 55\)](#).

### Evaluate the Model Deployed to SageMaker Hosting Services

To evaluate the model and use it in production, invoke the endpoint with the test dataset and check whether the inferences you get returns a target accuracy you want to achieve.

#### To evaluate the model

1. Set up the following function to predict each line of the test set. In the following example code, the `rows` argument is to specify the number of lines to predict at a time. You can change the value of it to perform a batch inference that fully utilizes the instance's hardware resource.

```
import numpy as np
def predict(data, rows=1000):
    split_array = np.array_split(data, int(data.shape[0] / float(rows) + 1))
```

```

predictions = ''
for array in split_array:
    predictions = ','.join([predictions,
xgb_predictor.predict(array).decode('utf-8')])
return np.fromstring(predictions[1:], sep=',')

```

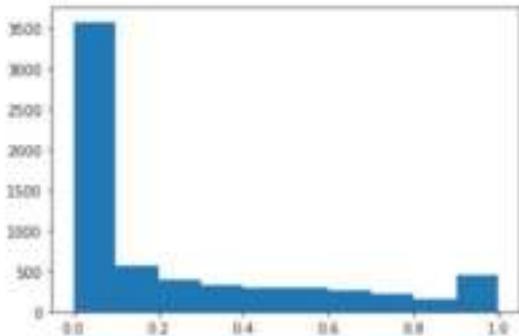
- Run the following code to make predictions of the test dataset and plot a histogram. You need to take only the feature columns of the test dataset, excluding the 0th column for the actual values.

```

import matplotlib.pyplot as plt

predictions=predict(test.to_numpy()[:,1:])
plt.hist(predictions)
plt.show()

```



- The predicted values are float type. To determine True or False based on the float values, you need to set a cutoff value. As shown in the following example code, use the Scikit-learn library to return the output confusion metrics and classification report with a cutoff of 0.5.

```

import sklearn

cutoff=0.5
print(sklearn.metrics.confusion_matrix(test.iloc[:, 0], np.where(predictions > cutoff,
1, 0)))
print(sklearn.metrics.classification_report(test.iloc[:, 0], np.where(predictions >
cutoff, 1, 0)))

```

This should return the following confusion matrix:

	precision	recall	f1-score	support
0	0.91	0.93	0.92	5826
1	0.74	0.68	0.71	1487
accuracy			0.87	6513
macro avg	0.82	0.80	0.81	6513
weighted avg	0.87	0.87	0.87	6513

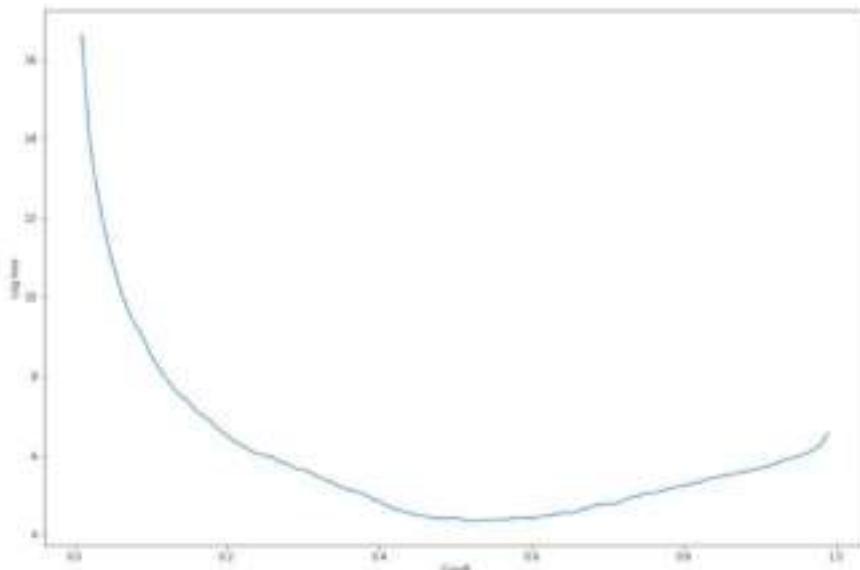
- To find the best cutoff with the given test set, compute the log loss function of the logistic regression. The log loss function is defined as the negative log-likelihood of a logistic model that returns prediction probabilities for its ground truth labels. The following example code numerically and iteratively calculates the log loss values  $-(y \log(p) + (1-y) \log(1-p))$ , where  $y$  is the true label and  $p$  is a probability estimate of the corresponding test sample. It returns a log loss versus cutoff graph.

```
import matplotlib.pyplot as plt

cutoffs = np.arange(0.01, 1, 0.01)
log_loss = []
for c in cutoffs:
    log_loss.append(
        sklearn.metrics.log_loss(test.iloc[:, 0], np.where(predictions > c, 1, 0))
    )

plt.figure(figsize=(15,10))
plt.plot(cutoffs, log_loss)
plt.xlabel("Cutoff")
plt.ylabel("Log loss")
plt.show()
```

This should return the following log loss curve.



- Find the minimum points of the error curve using the NumPy `argmin` and `min` functions:

```
print(
    'Log loss is minimized at a cutoff of ', cutoffs[np.argmin(log_loss)],
    ', and the log loss value at the minimum is ', np.min(log_loss)
)
```

This should return: Log loss is minimized at a cutoff of 0.53, and the log loss value at the minimum is 4.348539186773897.

Instead of computing and minimizing the log loss function, you can estimate a cost function as an alternative. For example, if you want to train a model to perform a binary classification for a business problem such as a customer churn prediction problem, you can set weights to the elements of confusion matrix and calculate the cost function accordingly.

You have now trained, deployed, and evaluated your first model in SageMaker.

**Tip**

To monitor model quality, data quality, and bias drift, use Amazon SageMaker Model Monitor and SageMaker Clarify. To learn more, see [Amazon SageMaker Model Monitor](#), [Monitor Data Quality](#), [Monitor Model Quality](#), [Monitor Bias Drift](#), and [Monitor Feature Attribution Drift](#).

**Tip**

To get human review of low confidence ML predictions or a random sample of predictions, use Amazon Augmented AI human review workflows. For more information, see [Using Amazon Augmented AI for Human Review](#).

## Step 7: Clean Up

To avoid incurring unnecessary charges, use the Amazon Web Services Management Console to delete the endpoints and resources that you created while running the exercises.

**Note**

Training jobs and logs cannot be deleted and are retained indefinitely.

**Note**

If you plan to explore other exercises in this guide, you might want to keep some of these resources, such as your notebook instance, S3 bucket, and IAM role.

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/> and delete the following resources:
  - The endpoint. Deleting the endpoint also deletes the ML compute instance or instances that support it.
    1. Under **Inference**, choose **Endpoints**.
    2. Choose the endpoint that you created in the example, choose **Actions**, and then choose **Delete**.
  - The endpoint configuration.
    1. Under **Inference**, choose **Endpoint configurations**.
    2. Choose the endpoint configuration that you created in the example, choose **Actions**, and then choose **Delete**.
  - The model.
    1. Under **Inference**, choose **Models**.
    2. Choose the model that you created in the example, choose **Actions**, and then choose **Delete**.
  - The notebook instance. Before deleting the notebook instance, stop it.
    1. Under **Notebook**, choose **Notebook instances**.
    2. Choose the notebook instance that you created in the example, choose **Actions**, and then choose **Stop**. The notebook instance takes several minutes to stop. When the **Status** changes to **Stopped**, move on to the next step.
    3. Choose **Actions**, and then choose **Delete**.
2. Open the Amazon S3 console at <https://console.amazonaws.cn/s3/>, and then delete the bucket that you created for storing model artifacts and the training dataset.
3. Open the Amazon CloudWatch console at <https://console.amazonaws.cn/cloudwatch/>, and then delete all of the log groups that have names starting with /aws/sagemaker/.

# Amazon SageMaker Studio

Amazon SageMaker Studio is a web-based, integrated development environment (IDE) for machine learning that lets you build, train, debug, deploy, and monitor your machine learning models. SageMaker Studio provides all the tools you need to take your models from experimentation to production while boosting your productivity. In a single unified visual interface, customers can perform the following tasks:

- Write and execute code in Jupyter notebooks
- Build and train machine learning models
- Deploy the models and monitor the performance of their predictions
- Track and debug the machine learning experiments

For information on the onboarding steps to sign in to SageMaker Studio, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).

For the Amazon Regions supported by SageMaker Studio, see [Supported Regions and Quotas \(p. 32\)](#).

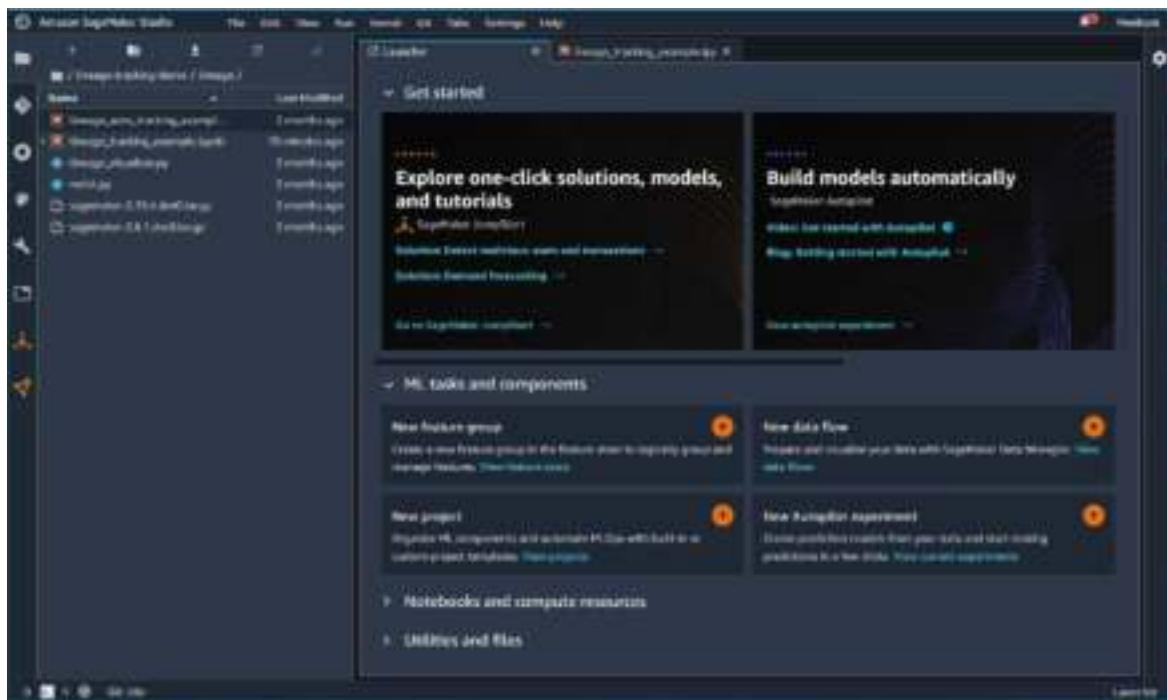
## Topics

- [Amazon SageMaker Studio UI Overview \(p. 69\)](#)
- [Use the Amazon SageMaker Studio Launcher \(p. 72\)](#)
- [Studio Entity Status \(p. 74\)](#)
- [Use Amazon SageMaker Studio Notebooks \(p. 76\)](#)
- [Bring your own SageMaker image \(p. 95\)](#)
- [Set Up a Connection to an Amazon EMR Cluster \(p. 110\)](#)
- [Perform Common Tasks in Amazon SageMaker Studio \(p. 113\)](#)
- [Amazon SageMaker Studio Pricing \(p. 119\)](#)
- [Troubleshooting Amazon SageMaker Studio \(p. 119\)](#)

## Amazon SageMaker Studio UI Overview

Amazon SageMaker Studio extends the JupyterLab interface. Previous users of JupyterLab will notice the similarity of the user interface, including the workspace. Studio adds many additions to the interface. The most prominent additions are detailed in the following sections. For an overview of the basic JupyterLab interface, see [The JupyterLab Interface](#).

The following image shows SageMaker Studio with the file browser open and the Studio Launcher displayed.



At the top of the screen is the *menu bar*. At the left of the screen is the *left sidebar* which contains icons to open file browsers, resource browsers, and tools. At the right of the screen is the *right sidebar*, represented by the **Settings** icon (⚙️), which displays contextual property settings when open. At the bottom of the screen is the *status bar*.

Above the **Settings** icon, there's a button to provide feedback about your experiences with SageMaker Studio.

To the left of the **Feedback** button there's the notification icon. Choose the icon to view notifications from Studio such as new Studio versions and new SageMaker features. To update to a new version of Studio, see [Update SageMaker Studio and Studio Apps \(p. 117\)](#).

The main work area is divided horizontally into two panes. The left pane is the *file and resource browser*. The right pane contains one or more tabs for resources such as notebooks, terminals, metrics, and graphs.

### Topics

- [Left sidebar \(p. 70\)](#)
- [File and resource browser \(p. 71\)](#)
- [Main work area \(p. 72\)](#)
- [Settings \(p. 72\)](#)

## Left sidebar

The left sidebar includes the following icons. When you hover over an icon, a tooltip displays the icon name. When you choose an icon, the file and resource browser displays the described functionality. For hierarchical entries, a selectable breadcrumb at the top of the browser shows your location in the hierarchy.

Icon	Description
	<p><b>File Browser</b></p> <p>Choose the <b>Upload Files</b> icon () to add files to Studio.</p> <p>Double-click a file to open the file in a new tab.</p> <p>To have adjacent files open, choose a tab that contains a notebook, Python, or text file, then choose <b>New View for File</b>.</p> <p>Choose the plus (+) sign on the menu at the top of the file browser to open the Studio Launcher.</p>
	<p><b>Running Terminals and Kernels</b></p> <p>For more information, see <a href="#">Shut Down Resources (p. 90)</a>.</p>
	<p><b>Git</b></p> <p>You can connect to a Git repository and then access a full range of Git tools and operations. For more information, see <a href="#">Clone a Git Repository in SageMaker Studio (p. 114)</a>.</p>
	<p><b>Commands (Ctrl + Shift + C)</b></p> <p>The majority of the menu commands are available here.</p>
	<p><b>Notebook Tools</b></p> <p>You can access a notebook's metadata through the <b>Advanced Tools</b> section. This icon is displayed only when a notebook is open.</p>
	<p><b>Open Tabs</b></p> <p>Provides a list of open tabs, which is useful if you have multiple open tabs.</p>
	<p><b>SageMaker Jumpstart</b></p> <p>Provides a list of solutions, model endpoints, or training jobs created with SageMaker Jumpstart.</p>
	<p><b>SageMaker Components and registries</b></p> <p>Provides a list of projects, data wrangler flows, pipelines, experiments, trials, models, or endpoints, or access to the feature store.</p>

## File and resource browser

The file and resource browser displays lists of your notebooks, experiments, trials, trial components, and endpoints. On the menu at the top of the file browser, choose the plus (+) sign to open the Studio Launcher. The Launcher allows you to create a notebook, launch a Python interactive shell, or open a terminal.

## Main work area

The main work area consists of multiple tabs that contain your open notebooks and terminals, and detailed information about your experiments and endpoints. One commonly used tab is the **Trial Component List**. This list is referred to as the *Leaderboard* because it's where you can compare experiments and trials. For more information, see [View and Compare Amazon SageMaker Experiments, Trials, and Trial Components \(p. 1558\)](#).

## Settings

The settings pane allows you to adjust table and chart properties. By default, the pane is hidden on the far right of the screen. To open the pane, choose the **Settings** icon (⚙) on the top right of the screen.

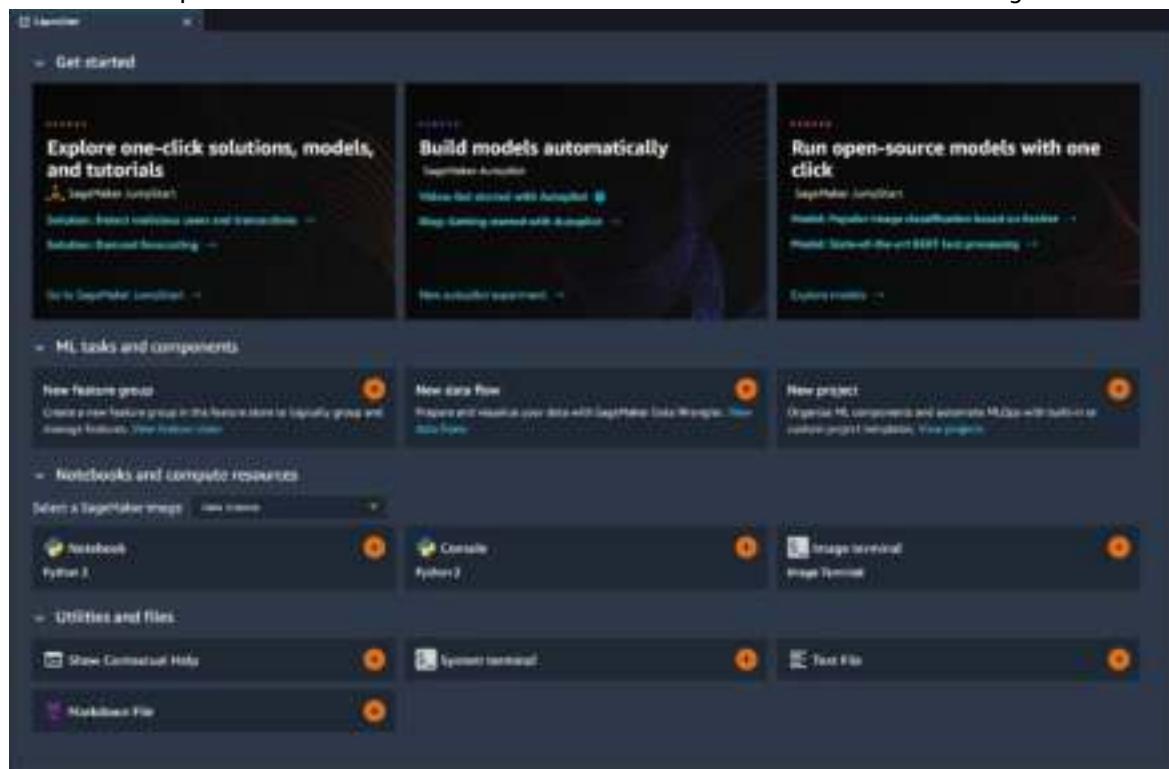
# Use the Amazon SageMaker Studio Launcher

You can use the Amazon SageMaker Studio Launcher to create notebooks and text files, and launch terminals and interactive Python shells.

You can open Studio Launcher in any of the following ways:

- Choose **Amazon SageMaker Studio** at the top-left of Studio.
- Use the keyboard shortcut **Ctrl + Shift + L**.
- From the Studio menu, choose **File** and then choose **New Launcher**.
- If the Studio file browser is open, choose the plus (+) sign on the Studio file browser menu.

The Launcher opens in a new tab in Studio. Your screen should look similar to the following:



The Launcher consists of following sections:

- **Get started** – Provides material to get started using SageMaker Studio, such as videos and tutorials, and one-click solutions for machine learning problems.
- **ML tasks and components** – Create machine learning tasks and components, such as new feature groups, data flows, and projects.
- **Notebooks and compute resources** – Create a notebook, open an image terminal, or open a Python console.
- **Utilities and files** – Show contextual help from a notebook, create files, or open a system terminal.

#### Topics

- [Notebooks and compute resources \(p. 73\)](#)
- [Utilities and files \(p. 73\)](#)

## Notebooks and compute resources

To create or launch an item, choose the SageMaker image that you want the item to run in from the **SageMaker image** dropdown menu. Next, choose the item. When you choose an item from this section, you might incur additional usage charges. For more information, see [Usage Metering \(p. 92\)](#).

The following items are available:

- **Notebook**

Launches the notebook in a kernel session on the chosen SageMaker image. For more information, see [Change a Kernel \(p. 87\)](#).

Creates the notebook in the folder that you have currently selected in the file browser. To view the file browser, in the left sidebar of Studio, choose the **File Browser** icon ().

- **Console**

Launches the shell in a kernel session on the chosen SageMaker image.

Opens the shell in the folder that you have currently selected in the file browser.

- **Image terminal**

Launches the terminal in a terminal session on the chosen SageMaker image.

Opens the terminal in the root folder for the user (as shown by the Home folder in the file browser).

## Utilities and files

Items in this section run in the context of SageMaker Studio and don't incur usage charges.

The following items are available:

- **Show Contextual Help**

Opens a new tab that displays contextual help for functions in a Studio notebook. To display the help, choose a function in an active notebook. To make it easier to see the help in context, drag the help tab so that it's adjacent to the notebook tab. To open the help tab from within a notebook, press **Ctrl + I**.

The following screenshot shows the contextual help for the `Experiment.create` method.

A screenshot of the Amazon SageMaker Studio interface. A code editor window titled "mnist-handwritten-digits-classifier.ipynb" is open, showing Python code. A tooltip or contextual help box is overlaid on the screen, highlighting the `Experiment.create` method. The tooltip contains the following information:

- Signature:** `Experiment.create(  
 experiment_name<str>,  
 description<str>,  
 sagemaker_boto_client<SageMaker.Client>)`
- Description:** Create a new experiment in SageMaker and return an ``Experiment`` object.
- Args:**
  - `experiment_name`: (str) - Name of the experiment. Must be unique. Required.
  - `experiment_description`: (str, optional) - Description of the experiment.
  - `sagemaker_boto_client`: (SageMaker.Client, optional) - Boto3 client for SageMaker. If not supplied, a default boto3 client will be created and used.
- Returns:** `sagemaker.experiments.Experiment`: A SageMaker ``Experiment`` object
- File:** /opt/conda/lib/python3.7/site-packages/sagemaker/experiments/experiment.py
- Type:** method

- **System terminal**

Opens a bash shell in the root folder for the user (as shown by the Home folder in the file browser).

- **Text File and Markdown File**

Creates a file of the associated type in the folder that you have currently selected in the file browser.

To view the file browser, in the left sidebar, choose the **File Browser** icon (  ).

## Studio Entity Status

Amazon SageMaker Studio creates the following entities for various purposes.

- **Domain:** A SageMaker Studio domain consists of an associated Amazon Elastic File System (Amazon EFS) volume; a list of authorized users; and a variety of security, application, policy, and Amazon Virtual Private Cloud (Amazon VPC) configurations. An Amazon account is limited to one domain per Region. Users within a domain can share notebook files and other artifacts with each other.
- **UserProfile:** A user profile represents a single user within a domain, and is the main way to reference a user for the purposes of sharing, reporting, and other user-oriented features. This entity is created when a user onboards to SageMaker Studio.
- **App:** An app represents an application that supports the reading and execution experience of the user's notebooks, terminals, and consoles. Apps can be either a JupyterServer or a KernelGateway. This operation is automatically invoked by SageMaker Studio upon access to the associated Domain, and when new kernel configurations are selected by the user. A user may have multiple Apps active simultaneously.

The following tables describe the status values for the Domain, UserProfile, and App entities. Where applicable, they also give troubleshooting steps.

#### **Domain status values**

<b>Value</b>	<b>Description</b>
Pending	Ongoing creation of Domain.
InService	Successful creation of Domain.
Updating	Ongoing update of Domain.
Deleting	Ongoing deletion of Domain.
Failed	Unsuccessful creation of Domain. Call the <code>DescribeDomain</code> API to see the failure reason for Domain creation. Delete the failed Domain and recreate the Domain after fixing the error mentioned in <code>FailureReason</code> .
Update_Failed	Unsuccessful update of Domain. Call the <code>DescribeDomain</code> API to see the failure reason for Domain update. Call the <code>UpdateDomain</code> API after fixing the error mentioned in <code>FailureReason</code> .
Delete_Failed	Unsuccessful deletion of Domain. Call the <code>DescribeDomain</code> API to see the failure reason for Domain deletion. Because deletion failed, you might have some resources that are still running, but you cannot use or update the Domain. Call the <code>DeleteDomain</code> API again after fixing the error mentioned in <code>FailureReason</code> .

#### **UserProfile status values**

<b>Value</b>	<b>Description</b>
Pending	Ongoing creation of UserProfile.
InService	Successful creation of UserProfile.
Updating	Ongoing update of UserProfile.
Deleting	Ongoing deletion of UserProfile.
Failed	Unsuccessful creation of UserProfile. Unsuccessful creation of UserProfile. Call the <code>DescribeUserProfile</code> API to see the failure reason for UserProfile creation. Delete the failed UserProfile and recreate it after fixing the error mentioned in <code>FailureReason</code> .
Update_Failed	Unsuccessful update of UserProfile. Call the <code>DescribeUserProfile</code> API to see the failure reason for UserProfile update. Call the <code>UpdateUserProfile</code> API again after fixing the error mentioned in <code>FailureReason</code> .

Value	Description
Delete_Failed	Unsuccessful deletion of UserProfile. Call the <code>DescribeUserProfile</code> API to see the failure reason for UserProfile deletion. Because deletion failed, you might have some resources that are still running, but you cannot use or update the UserProfile. Call the <code>DeleteUserProfile</code> API again after fixing the error mentioned in <code>FailureReason</code> .

#### App status values

Value	Description
Pending	Ongoing creation of App.
InService	Successful creation of App.
Deleting	Ongoing deletion of App.
Failed	Unsuccessful creation of App. Call the <code>DescribeApp</code> API to see the failure reason for App creation. Call the <code>CreateApp</code> API again after fixing the error mentioned in <code>FailureReason</code> .
Deleted	Successful deletion of App.

## Use Amazon SageMaker Studio Notebooks

Amazon SageMaker Studio notebooks are collaborative notebooks that you can launch quickly because you don't need to set up compute instances and file storage beforehand. A set of instance types, known as *Fast launch* types are designed to launch in under two minutes. SageMaker Studio notebooks provide persistent storage, which enables you to view and share notebooks even if the instances that the notebooks run on are shut down.

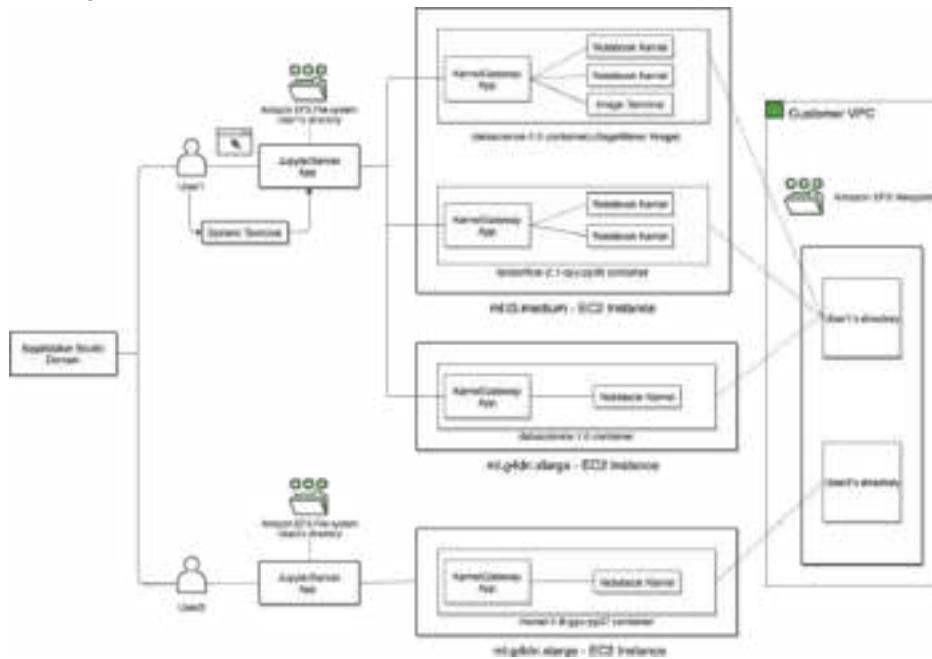
You can share your notebooks with others, so that they can easily reproduce your results and collaborate while building models and exploring your data. You provide access to a read-only copy of the notebook through a secure URL. Dependencies for your notebook are included in the notebook's metadata. When your colleagues copy the notebook, it opens in the same environment as the original notebook.

A SageMaker Studio notebook runs in an environment defined by the following:

- EC2 instance type – The hardware configuration the notebook runs on. The configuration includes the number and type of processors (vCPU and GPU), and the amount and type of memory. The instance type determines the pricing rate.
- SageMaker image – A container image that is compatible with SageMaker Studio. The image consists of the kernels, language packages, and other files required to run a notebook in Studio. There can be multiple images in an instance. For more information, see [Bring your own SageMaker image \(p. 95\)](#).
- KernelGateway app – A SageMaker image runs as a KernelGateway app. The app provides access to the kernels in the image. There is a one-to-one correspondence between a SageMaker image and a SageMaker app.
- Kernel – The process that inspects and runs the code contained in the notebook. A kernel is defined by a *kernel spec* in the image. There can be multiple kernels in an image.

You can change any of these resources from within the notebook.

The following diagram outlines how a notebook kernel runs in relation to the KernelGateway App, User, and SageMaker Studio Domain.



Sample SageMaker Studio notebooks are available in the [aws\\_sagemaker\\_studio](#) folder of the [Amazon SageMaker example GitHub repository](#). Each notebook comes with the necessary SageMaker image that opens the notebook with the appropriate kernel.

We recommend that you familiarize yourself with the SageMaker Studio interface and the Studio notebook toolbar before creating or using a Studio notebook. For more information, see [Amazon SageMaker Studio UI Overview \(p. 69\)](#) and [Use the SageMaker Studio Notebook Toolbar \(p. 80\)](#).

### Topics

- [How Are Amazon SageMaker Studio Notebooks Different from Notebook Instances? \(p. 77\)](#)
- [Get Started \(p. 78\)](#)
- [Create or Open an Amazon SageMaker Studio Notebook \(p. 79\)](#)
- [Use the SageMaker Studio Notebook Toolbar \(p. 80\)](#)
- [Share and Use a Amazon SageMaker Studio Notebook \(p. 82\)](#)
- [Get Notebook and App Metadata \(p. 84\)](#)
- [Get Notebook Differences \(p. 85\)](#)
- [Manage Resources \(p. 86\)](#)
- [Usage Metering \(p. 92\)](#)
- [Available Resources \(p. 92\)](#)

## How Are Amazon SageMaker Studio Notebooks Different from Notebook Instances?

When you're starting a new notebook, we recommend that you create the notebook in Amazon SageMaker Studio instead of launching a notebook instance from the Amazon SageMaker console. There are many benefits to using a SageMaker Studio notebook, including the following:

- Starting a Studio notebook is faster than launching an instance-based notebook. Typically, it is 5-10 times faster than instance-based notebooks.
- Notebook sharing is an integrated feature in SageMaker Studio. Users can generate a shareable link that reproduces the notebook code and also the SageMaker image required to execute it, in just a few clicks.
- SageMaker Studio notebooks come pre-installed with the latest [Amazon SageMaker Python SDK](#).
- SageMaker Studio notebooks are accessed from within Studio. This enables you to build, train, debug, track, and monitor your models without leaving Studio.
- Each member of a Studio team gets their own home directory to store their notebooks and other files. The directory is automatically mounted onto all instances and kernels as they're started, so their notebooks and other files are always available. The home directories are stored in Amazon Elastic File System (Amazon EFS) so that you can access them from other services.
- When using Amazon Web Services SSO, you use your SSO credentials through a unique URL to directly access SageMaker Studio. You don't have to interact with the Amazon Web Services Management Console to run your notebooks.
- Studio notebooks are equipped with a set of predefined SageMaker image settings to get you started faster.

**Note**

Studio notebooks don't support *local mode*. However, you can use a notebook instance to train a sample of your dataset locally, and then use the same code in a Studio notebook to train on the full dataset.

When you open a notebook in SageMaker Studio, the view is an extension of the JupyterLab interface. The primary features are the same, so you'll find the typical features of a Jupyter notebook and JupyterLab. For more information about the Studio interface, see [Amazon SageMaker Studio UI Overview \(p. 69\)](#).

## Get Started

To get started, you or your organization's administrator need to complete the Amazon SageMaker Studio onboarding process. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).

You can access an SageMaker Studio notebook in any of the following ways:

- You receive an email invitation to access Studio through your organization's Amazon Web Services SSO account, which includes a direct link to login to Studio without having to use the Amazon SageMaker console. You can proceed to the [the section called "Next Steps" \(p. 79\)](#).
- You receive a link to a shared Studio notebook, which includes a direct link to log in to Studio without having to use the SageMaker console. You can proceed to the [the section called "Next Steps" \(p. 79\)](#).
- You onboard to Studio and then log in to the SageMaker console. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).

## Log In from the Amazon SageMaker console

### To log in from the SageMaker console

1. Onboard to Amazon SageMaker Studio. If you've already onboarded, skip to the next step.
2. Open the SageMaker [console](#).
3. Choose **Amazon SageMaker Studio**.
4. The Amazon SageMaker Studio Control Panel opens.
5. In the Amazon SageMaker Studio Control Panel, you'll see a list of user names.

Next to your user name, choose **Open Studio**.

## Next Steps

Now that you're in Studio, you can try any of the following options:

- Create a SageMaker Studio notebook – Continue to the next section.
- Familiarize yourself with the SageMaker Studio interface – See [Amazon SageMaker Studio UI Overview \(p. 69\)](#).
- Follow a Studio end-to-end tutorial – See [Amazon SageMaker Studio Tour \(p. 51\)](#).

## Create or Open an Amazon SageMaker Studio Notebook

When you create a notebook in Amazon SageMaker Studio or open a non-shared notebook in Studio for the first time, you have to select a SageMaker image and kernel for the notebook. SageMaker launches the notebook on a default instance of a type based on the chosen SageMaker image. For CPU based images, the default instance type is `ml.t3.medium` (available as part of the [Amazon Free Tier](#)). For GPU based images, the default instance type is `ml.g4dn.xlarge`.

If you create or open additional notebooks that use the same instance type, whether or not the notebooks use the same kernel, the notebooks run on the same instance of that instance type.

After a notebook is launched, you can change its instance type, and SageMaker image and kernel from within the notebook. For more information, see [Change an Instance Type \(p. 86\)](#) and [Change a Kernel \(p. 87\)](#).

You can have only one instance of each instance type. Each instance can have multiple SageMaker images running on it. Each SageMaker image can run multiple kernels or terminal instances.

Billing occurs per instance and starts when the first instance of a given instance type is launched. If you want to create or open a notebook without the risk of incurring charges, open the notebook from the **File** menu and choose **No Kernel** from the **Select Kernel** dialog. You can read and edit a notebook without a running kernel but you can't run cells.

Billing ends when the SageMaker image for the instance is shut down. For more information, see [Usage Metering \(p. 92\)](#).

For information on shutting down the notebook, see [Shut Down Resources \(p. 90\)](#).

### Topics

- [Open a Studio notebook \(p. 79\)](#)
- [Create a Notebook from the File Menu \(p. 80\)](#)
- [Create a Notebook from the Launcher \(p. 80\)](#)

## Open a Studio notebook

SageMaker Studio can only open notebooks listed in the Studio file browser. For instructions on uploading a notebook to the file browser, see [Upload Files to SageMaker Studio \(p. 114\)](#) or [Clone a Git Repository in SageMaker Studio \(p. 114\)](#).

### To open a notebook

1. In the left sidebar, choose the **File Browser** icon () to display the file browser.
2. Browse to a notebook file and double-click it to open the notebook in a new tab.

## Create a Notebook from the File Menu

### To create a notebook from the File menu

1. From the Studio menu, choose **File**, choose **New**, and then choose **Notebook**.
2. On the **Select Kernel** dialog, to use the default kernel, **Python 3 (Data Science)**, choose **Select**. Otherwise, use the dropdown menu to select a different kernel.

For a list of the available kernels, see [Available Amazon SageMaker Kernels \(p. 94\)](#).

## Create a Notebook from the Launcher

### To create a notebook from the Launcher

1. To open the Launcher, use the keyboard shortcut **Ctrl + Shift + L**. Alternatively, from the File Browser, choose the plus (+) sign on the left of the menu.
2. On the Launcher, keep the default SageMaker image, **Data Science**, or use the dropdown menu to select a different image.
3. Under **Notebook**, choose **Python3**.

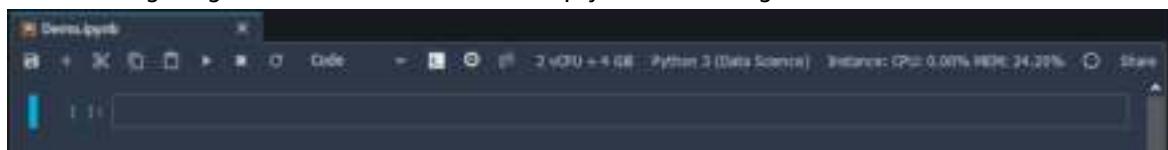
For a list of the available images, see [Available Amazon SageMaker Images \(p. 94\)](#).

After you choose the kernel or image, your notebook launches and opens in a new Studio tab. To view the notebook's kernel session, in the left sidebar, choose the **Running Terminals, Kernels, and Images** icon (  ). You can stop the notebook's kernel session from this view.

## Use the SageMaker Studio Notebook Toolbar

Amazon SageMaker Studio notebooks extend the JupyterLab interface. For an overview of the basic JupyterLab interface, see [The JupyterLab Interface](#).

The following image shows the toolbar and an empty cell from a SageMaker Studio notebook.

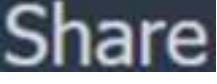


When you pause on a toolbar icon, a tooltip displays the icon function. Additional notebook commands are found in the Studio main menu. For a list of available notebook commands and shortcuts, in the

left sidebar of Studio, choose the **Commands** icon (  ), and then scroll to the **NOTEBOOK CELL OPERATIONS** and **NOTEBOOK OPERATIONS** sections. The toolbar includes the following icons:

Icon	Description
	<b>Save and checkpoint</b>  Saves the notebook and updates the checkpoint file. For more information, see <a href="#">Get the Difference Between the Last Checkpoint (p. 85)</a> .

Icon	Description
	<b>Insert cell</b> Inserts a code cell below the current cell. The current cell is noted by the blue vertical marker in the left margin.
	<b>Cut, copy, and paste cells</b> Cuts, copies, and pastes the selected cells.
	<b>Run cells</b> Runs the selected cells and then makes the cell that follows the last selected cell the new selected cell.
	<b>Interrupt kernel</b> Interrupts the kernel, which cancels the currently running operation. The kernel remains active.
	<b>Restart kernel</b> Restarts the kernel. Variables are reset. Unsaved information is not affected.
	<b>Cell type</b> Displays or changes the current cell type. The cell types are: <ul style="list-style-type: none"> <li>Code – Code that the kernel runs.</li> <li>Markdown – Text rendered as markdown.</li> <li>Raw – Content, including Markdown markup, that's displayed as text.</li> </ul>
	<b>Launch terminal</b> Launches a terminal in the SageMaker image hosting the notebook. For an example, see <a href="#">Get App Metadata (p. 84)</a> .
	<b>Checkpoint diff</b> Opens a new tab that displays the difference between the notebook and the checkpoint file. For more information, see <a href="#">Get the Difference Between the Last Checkpoint (p. 85)</a> .
	<b>Git diff</b> Only enabled if the notebook is opened from a Git repository. Opens a new tab that displays the difference between the notebook and the last Git commit. For more information, see <a href="#">Get the Difference Between the Last Commit (p. 86)</a> .

Icon	Description
<b>2 vCPU + 4 GiB</b>	<p><b>Instance type</b></p> <p>Displays or changes the instance type the notebook runs in. The format is as follows:</p> <pre>number of vCPUs + amount of memory + number of GPUs</pre> <p>Unknown indicates the notebook was opened without specifying a kernel. The notebook runs on the SageMaker Studio instance and doesn't accrue runtime charges. You can't assign the notebook to an instance type. You must specify a kernel and then Studio assigns the notebook to a default type.</p> <p>For more information, see <a href="#">Create or Open an Amazon SageMaker Studio Notebook (p. 79)</a> and <a href="#">Change an Instance Type (p. 86)</a>.</p>
<b>Python 3 (Data Science)</b>	<p><b>Kernel and SageMaker Image</b></p> <p>Displays or changes the kernel that processes the cells in the notebook. The format is as follows:</p> <pre>Kernel (SageMaker Image)</pre> <p>No Kernel indicates the notebook was opened without specifying a kernel. You can edit the notebook but you can't run any cells.</p> <p>For more information, see <a href="#">Change a Kernel (p. 87)</a>.</p>
<b>Kernel: CPU: 0.00% MEM: 2.81%</b>	<p><b>CPU and memory of the kernel or instance</b></p> <p>Displays the CPU usage and memory usage. Double-click to toggle between the current kernel and the current instance.</p>
	<p><b>Kernel busy status</b></p> <p>Displays the busy status of the kernel. When the edge of the circle and its interior are the same color, the kernel is busy. The kernel is busy when it is starting and when it is processing cells. Additional kernel states are displayed in the status bar at the bottom-left corner of SageMaker Studio.</p>
	<p><b>Share notebook</b></p> <p>Shares the notebook. For more information, see <a href="#">Share and Use a Amazon SageMaker Studio Notebook (p. 82)</a>.</p>

To select multiple cells, click in the left margin outside of a cell. Hold down the Shift key and use K or the Up key to select previous cells, or use J or the Down key to select following cells.

## Share and Use a Amazon SageMaker Studio Notebook

You can share your Amazon SageMaker Studio notebooks with your colleagues. The shared notebook is a copy. After you share your notebook, any changes you make to your original notebook aren't reflected in the shared notebook and any changes your colleague's make in their shared copies of the notebook

aren't reflected in your original notebook. If you want to share your latest version, you must create a new snapshot and then share it.

#### Topics

- [Share a Notebook \(p. 83\)](#)
- [Use a Shared Notebook \(p. 83\)](#)

## Share a Notebook

The following screenshot shows the menu from a Studio notebook.



#### To share a notebook

1. In the upper-right corner of the notebook, choose **Share**.
2. (Optional) In **Create shareable snapshot**, choose any of the following items:
  - **Include Git repo information** – Includes a link to the Git repository that contains the notebook. This enables you and your colleague to collaborate and contribute to the same Git repository.
  - **Include output** – Includes all notebook output that has been saved.

#### Note

If you're an Amazon Web Services SSO user and you don't see these options, your Amazon Web Services SSO administrator probably disabled the feature. Contact your administrator.

3. Choose **Create**.
4. After the snapshot is created, choose **Copy link** and then choose **Close**.
5. Share the link with your colleague.

After selecting your sharing options, you are provided with a URL. You can share this link with users that have access to Amazon SageMaker Studio. When the user opens the URL, they're prompted to log in using Amazon Web Services SSO or IAM authentication. This shared notebook becomes a copy, so changes made by the recipient will not be reproduced in your original notebook.

## Use a Shared Notebook

You use a shared notebook in the same way you would with a notebook that you created yourself. When you click a link to a shared notebook for the first time, a read-only version of the notebook opens. To edit the shared notebook, choose **Create a Copy**. This copies the shared notebook to your personal storage.

The copied notebook launches on an instance of the instance type and SageMaker image that the notebook was using when the sender shared it. If you aren't currently running an instance of the instance type, a new instance is started. Customization to the SageMaker image isn't shared. You can also inspect the notebook snapshot by choosing **Snapshot Details**.

The following are some important considerations about sharing and authentication:

- If you have an active session, you see a read-only view of the notebook until you choose **Create a Copy**.
- If you don't have an active session, you need to log in.

- If you use IAM to login, after you login, select your user profile then choose **Open SageMaker Studio**. Then you need to choose the link you were sent.
- If you use SSO to login, after you login the shared notebook is opened automatically in Studio.

## Get Notebook and App Metadata

You can access notebook metadata and App metadata using the Amazon SageMaker UI.

### Topics

- [Get Notebook Metadata \(p. 84\)](#)
- [Get App Metadata \(p. 84\)](#)

## Get Notebook Metadata

Jupyter notebooks contain optional metadata that you can access through the Amazon SageMaker UI.

### To view the notebook metadata

1. In the left sidebar, choose the **Notebook Tools** icon ().
2. Open the **Advanced Tools** section.

The metadata should look similar to the following.

```
{  
    "instance_type": "ml.t3.medium",  
    "kernelspec": {  
        "display_name": "Python 3 (Data Science)",  
        "language": "python",  
        "name": "python3__SAGEMAKER_INTERNAL__arn:aws:sagemaker:us-west-2:<acct-id>:image/  
data-science-1.0"  
    },  
    "language_info": {  
        "codemirror_mode": {  
            "name": "ipython",  
            "version": 3  
        },  
        "file_extension": ".py",  
        "mimetype": "text/x-python",  
        "name": "python",  
        "nbconvert_exporter": "python",  
        "pygments_lexer": "ipython3",  
        "version": "3.7.6"  
    }  
}
```

## Get App Metadata

When you create a notebook in Amazon SageMaker Studio, the App metadata is written to a file named `resource-metadata.json` in the folder `/opt/ml/metadata/`. You can get the App metadata by opening an Image terminal from within the notebook. The metadata gives you the following information, which includes the SageMaker image and instance type the notebook runs in:

- **AppType – KernelGateway**
- **DomainId – Same as the StudioID**

- **UserProfileName** – The profile name of the current user
- **ResourceArn** – The Amazon Resource Name (ARN) of the App, which includes the instance type
- **ResourceName** – The name of the SageMaker image

Additional metadata might be included for internal use by Studio and is subject to change.

#### To get the App metadata

1. In the center of the notebook menu, choose the **Launch Terminal** icon (terminal icon). This opens a terminal in the SageMaker image that the notebook runs in.
2. Run the following commands to display the contents of the `resource-metadata.json` file.

```
cd /opt/ml/metadata/  
cat resource-metadata.json
```

The file should look similar to the following.

```
{  
    "AppType": "KernelGateway",  
    "DomainId": "d-xxxxxxxxxxxxxx",  
    "UserProfileName": "profile-name",  
    "ResourceArn": "arn:aws:sagemaker:us-east-2:account-id:app/d-xxxxxxxxxxxxxx/profile-name/  
KernelGateway/datascience--1-0-ml-t3-medium",  
    "ResourceName": "datascience--1-0-ml"  
}
```

## Get Notebook Differences

You can display the difference between the current notebook and the last checkpoint or the last Git commit using the Amazon SageMaker UI.

The following screenshot shows the menu from a Studio notebook.



#### Topics

- [Get the Difference Between the Last Checkpoint \(p. 85\)](#)
- [Get the Difference Between the Last Commit \(p. 86\)](#)

## Get the Difference Between the Last Checkpoint

When you create a notebook, a hidden checkpoint file that matches the notebook is created. You can view changes between the notebook and the checkpoint file or revert the notebook to match the checkpoint file.

By default, a notebook is auto-saved every 120 seconds and also when you close the notebook. However, the checkpoint file isn't updated to match the notebook. To save the notebook and update the checkpoint file to match, you must choose the **Save notebook and create checkpoint** icon (disk icon) on the left of the notebook menu or use the **Ctrl + S** keyboard shortcut.

To view the changes between the notebook and the checkpoint file, choose the **Checkpoint diff** icon (  ) in the center of the notebook menu.

To revert the notebook to the checkpoint file, from the main Studio menu, choose **File** then **Revert Notebook to Checkpoint**.

## Get the Difference Between the Last Commit

If a notebook is opened from a Git repository, you can view the difference between the notebook and the last Git commit.

To view the changes in the notebook from the last Git commit, choose the **Git diff** icon (  ) in the center of the notebook menu.

## Manage Resources

You can change the instance type, and SageMaker image and kernel from within an Amazon SageMaker Studio notebook. To create a custom kernel to use with your notebooks, see [Bring your own SageMaker image \(p. 95\)](#).

### Topics

- [Change an Instance Type \(p. 86\)](#)
- [Change a Kernel \(p. 87\)](#)
- [Shut Down Resources \(p. 90\)](#)

## Change an Instance Type

When you open a new notebook for the first time, you are assigned a default Amazon Elastic Compute Cloud (Amazon EC2) instance type to run the notebook. When you open additional notebooks on the same instance type, the notebooks run on the same instance as the first notebook, even if the notebooks use different kernels.

You can change the instance type that your notebook runs on from within the notebook.

### Important

If you change the instance type, unsaved information and existing settings for the notebook are lost, and installed packages must be re-installed.

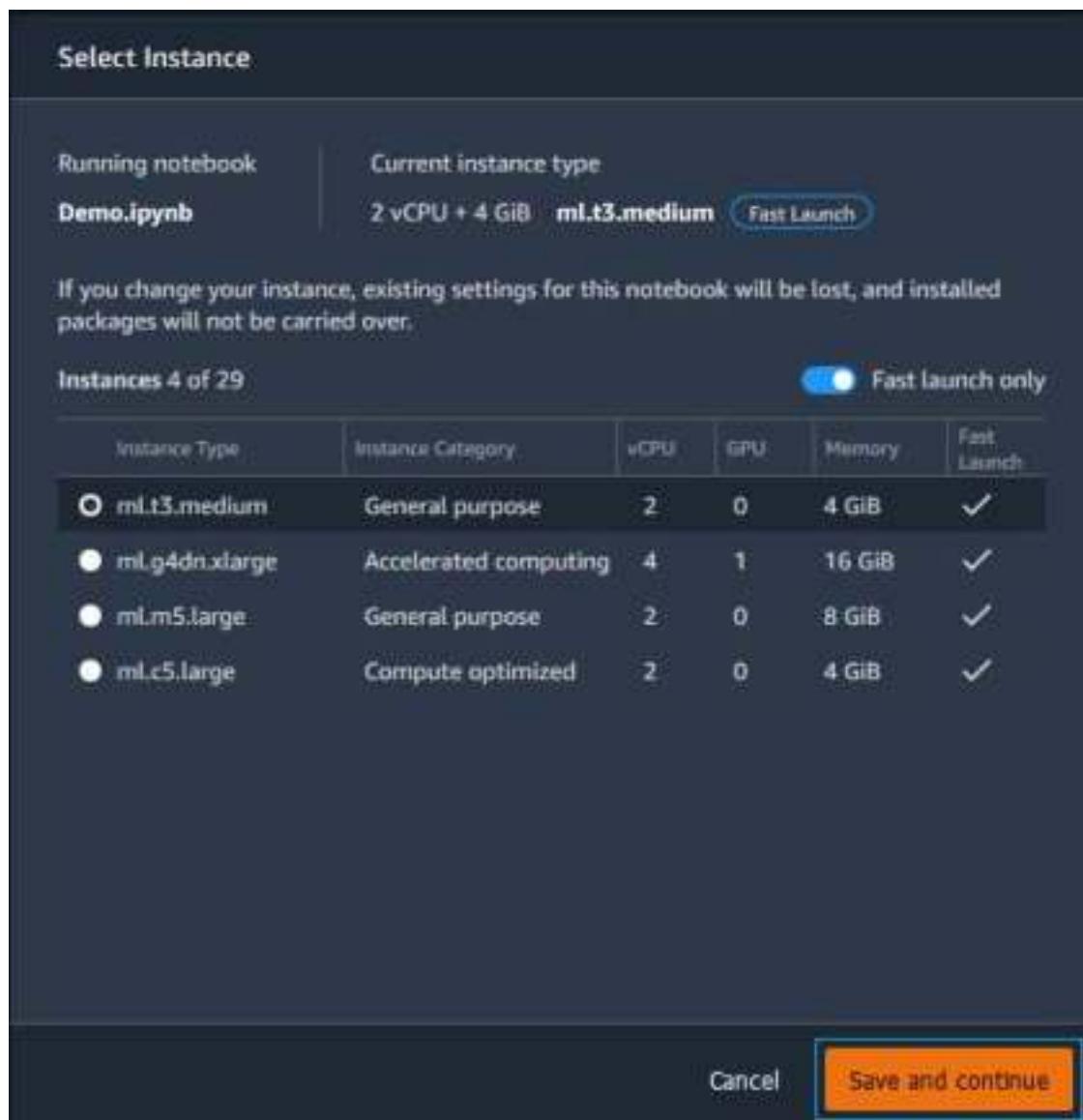
The previous instance type continues to run even if no kernel sessions or apps are active. You must explicitly stop the instance to stop accruing charges. To stop the instance, see [Shut Down Resources \(p. 90\)](#).

The following screenshot shows the menu from a Studio notebook. The processor and memory of the instance type powering the notebook are displayed as **2 vCPU + 4 GiB**.



### To change the instance type

1. Choose the instance type.
2. In **Select instance**, choose one of the fast launch instance types that are listed. Or to see all instance types, switch off **Fast launch only**. The list can be sorted by any column.



3. After choosing a type, choose **Save and continue**.
4. Wait for the new instance to become enabled, and then the new instance type information is displayed.

For a list of the available instance types, see [Available SageMaker Studio Instance Types \(p. 93\)](#).

## Change a Kernel

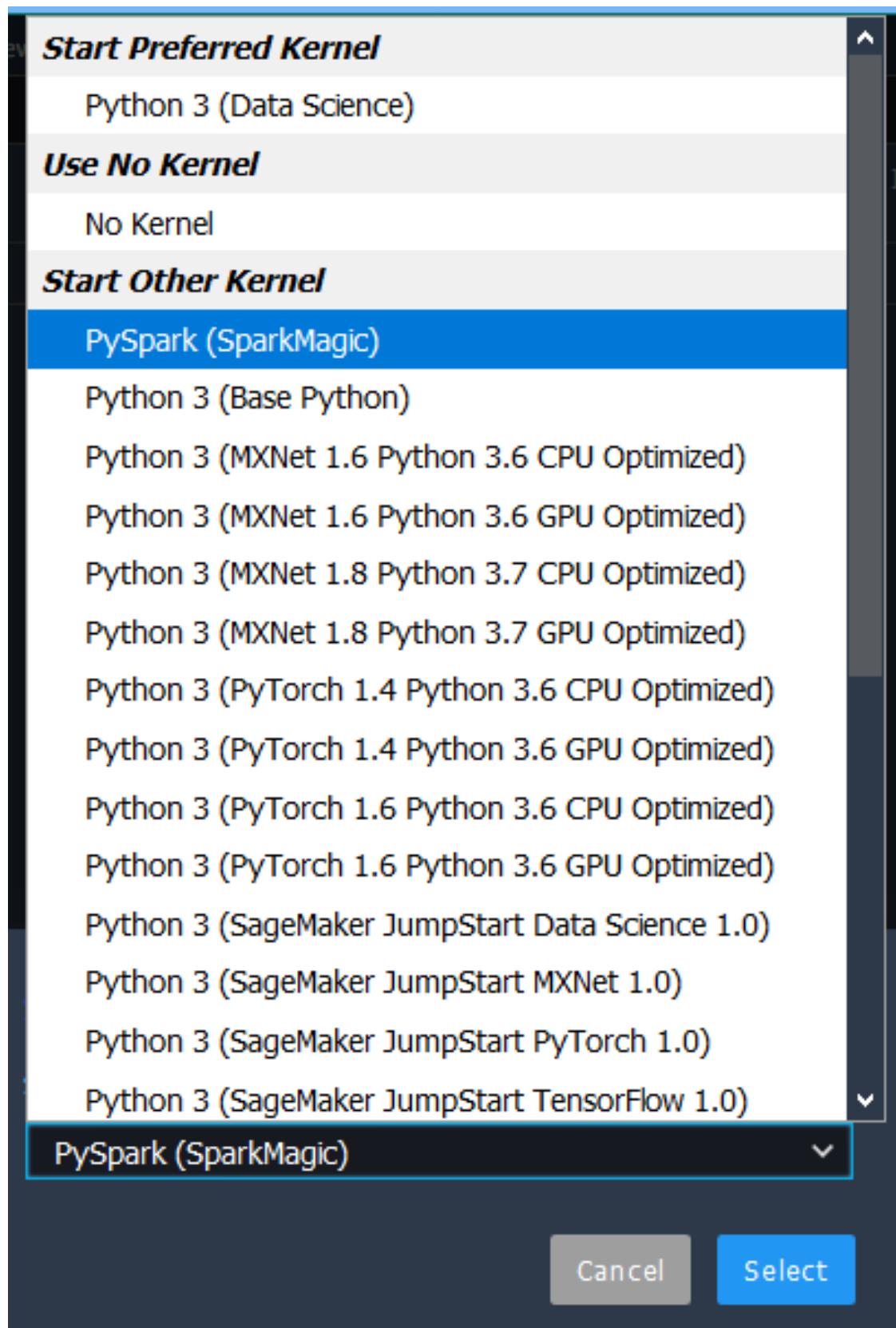
With Amazon SageMaker Studio notebooks, you can change the notebook's kernel from within the notebook.

The following screenshot shows the menu from a Studio notebook. The current SageMaker kernel and image are displayed as **Python 3 (Data Science)**, where Python 3 denotes the kernel and Data Science denotes the SageMaker image that contains the kernel. The color of the circle to the right indicates the kernel is idle or busy. The kernel is busy when the center and the edge of the circle are the same color.



**To change a notebook's kernel**

1. Choose the kernel name.
2. From the drop-down list, choose a kernel.



3. After choosing a kernel, choose **Select**.
4. Wait for the kernel's status to show as idle, which indicates the kernel has started.

For a list of available SageMaker kernels, see [Available Amazon SageMaker Kernels \(p. 94\)](#).

## Shut Down Resources

You can shut down individual resources, including notebooks, terminals, kernels, apps, and instances. You can also shut down all resources in one of these categories at the same time.

### Topics

- [Shut Down an Open Notebook \(p. 90\)](#)
- [Shut Down Resources \(p. 90\)](#)

## Shut Down an Open Notebook

You can shut down an open notebook from the Amazon SageMaker Studio **File** menu or from the Running Terminal and Kernels pane.

### Note

When you shut down a notebook, any unsaved information in the notebook is lost. The notebook is not deleted.

### To shut down an open notebook from the File menu

1. Optionally, save the notebook contents by choosing the **Disk** icon on the left of the notebook menu.
2. Choose **File** then **Close and Shutdown Notebook**.
3. Choose **OK**.

## Shut Down Resources

You can reach the **Running Terminals and Kernels** pane on the left side of Amazon SageMaker Studio



with the  icon. The **Running Terminals and Kernels** pane consists of four sections. Each section lists all the resources of that type. You can shut down each resource individually or shut down all the resources in a section at the same time.



When you choose to shut down all resources in a section, the following occurs:

- **RUNNING INSTANCES/RUNNING APPS** – All instances, apps, notebooks, kernel sessions, consoles/shells, and image terminals are shut down. System terminals aren't shut down. Choose this option to stop the accrual of all charges.
- **KERNEL SESSIONS** – All kernels, notebooks and consoles/shells are shut down.
- **TERMINAL SESSIONS** – All image terminals and system terminals are shut down.

### To shut down resources

1. In the left sidebar, choose the **Running Terminals and Kernels** icon ( ).
2. Do either of the following:
  - To shut down a specific resource, choose the **SHUT DOWN** icon ( ) on the same row as the resource.

For running instances, a confirmation dialog lists all the resources that will be shut down. For running apps, a confirmation dialog is displayed. Choose **Shut down all** to proceed.

#### Note

No confirmation dialog is displayed for kernel sessions or terminal sessions.

- To shut down all resources in a section, choose the X to the right of the section label. A confirmation dialog is displayed. Choose **Shut down all** to proceed.

## Usage Metering

There is no additional charge for using Amazon SageMaker Studio. The costs incurred for running Amazon SageMaker Studio notebooks, interactive shells, consoles, and terminals are based on Amazon Elastic Compute Cloud (Amazon EC2) instance usage.

When you run the following resources, you must choose a SageMaker image and kernel:

### From the Studio Launcher

- Notebook
- Interactive Shell
- Image Terminal

### From the File menu

- Notebook
- Console

When launched, the resource is run on an Amazon EC2 instance of an instance type based on the chosen SageMaker image and kernel. If an instance of that type was previously launched and is available, the resource is run on that instance.

For CPU based images, the default instance type is `m1.t3.medium`. For GPU based images, the default instance type is `m1.g4dn.xlarge`.

The costs incurred are based on the instance type. You are billed separately for each instance.

Metering starts when an instance is created. Metering ends when all the apps on the instance are shut down, or the instance is shut down. For information on how to shut down an instance, see [Shut Down Resources \(p. 90\)](#).

#### Important

You must shut down the instance to stop incurring charges. If you shut down the notebook running on the instance but don't shut down the instance, you will still incur charges.

When you open multiple notebooks on the same instance type, the notebooks run on the same instance even if they're using different kernels. You are billed only for the time that one instance is running.

You can change the instance type from within the notebook after you open it. For more information, see [Change an Instance Type \(p. 86\)](#).

For information about billing along with pricing examples, see [Amazon SageMaker Pricing](#).

## Available Resources

The following sections list the available resources for Amazon SageMaker Studio notebooks.

### Topics

- [Available SageMaker Studio Instance Types \(p. 93\)](#)
- [Available Amazon SageMaker Images \(p. 94\)](#)
- [Available Amazon SageMaker Kernels \(p. 94\)](#)

## Available SageMaker Studio Instance Types

The following Amazon Elastic Compute Cloud (Amazon EC2) instance types are available for use with SageMaker Studio notebooks.

For detailed information on which instance types fit your use case, and their performance capabilities, see [Amazon Elastic Compute Cloud Instance types](#).

For information on available Amazon SageMaker Notebook Instance types, see [CreateNotebookInstance](#).

### Note

For most use cases, you should use a ml.t3.medium. This is the default instance type for CPU-based SageMaker images, and is available as part of the [Amazon Free Tier](#).

>> *Fast launch* instances types are optimized to start in under two minutes.

### Default instance types

- CPU-based images: ml.t3.medium >> *Fast launch*
- GPU-based images: ml.g4dn.xlarge >> *Fast launch*

### General purpose (no GPUs)

- ml.t3.medium >> *Fast launch*
- ml.t3.large
- ml.t3.xlarge
- ml.t3.2xlarge
- ml.m5.large >> *Fast launch*
- ml.m5.xlarge
- ml.m5.2xlarge
- ml.m5.4xlarge
- ml.m5.8xlarge
- ml.m5.12xlarge
- ml.m5.16xlarge
- ml.m5.24xlarge

### Compute optimized (no GPUs)

- ml.c5.large >> *Fast launch*
- ml.c5.xlarge
- ml.c5.2xlarge
- ml.c5.4xlarge
- ml.c5.9xlarge
- ml.c5.12xlarge
- ml.c5.18xlarge
- ml.c5.24xlarge

### Accelerated computing (1+ GPUs)

- ml.p3.2xlarge
- ml.p3.8xlarge

- ml.p3.16xlarge
- ml.g4dn.xlarge >> *Fast launch*
- ml.g4dn.2xlarge
- ml.g4dn.4xlarge
- ml.g4dn.8xlarge
- ml.g4dn.12xlarge
- ml.g4dn.16xlarge

## Available Amazon SageMaker Images

The following SageMaker images are available in Amazon SageMaker Studio. SageMaker images contain the latest [Amazon SageMaker Python SDK](#) and the latest version of the kernel. The name in brackets ([ ]) is the resource identifier of the SageMaker image as specified in the Amazon Resource Name (ARN) for the SageMaker image. For more information, see [Deep Learning Containers Images](#).

- Data Science [datascience-1.0]

Data Science is a [Conda](#) image with the most commonly used Python packages and libraries, such as NumPy and SciKit Learn.

- Base Python [python-3.6]
- MXNet (optimized for CPU) [mxnet-1.6-cpu-py36]
- MXNet (optimized for GPU) [mxnet-1.6-gpu-py36]
- PyTorch (optimized for CPU) [pytorch-1.4-cpu-py36]
- PyTorch (optimized for GPU) [pytorch-1.4-gpu-py36]
- TensorFlow (optimized for CPU) [tensorflow-1.15-cpu-py36]
- TensorFlow (optimized for GPU) [tensorflow-1.15-gpu-py36]
- TensorFlow 2 (optimized for CPU) [tensorflow-2.1-cpu-py36]
- TensorFlow 2 (optimized for GPU) [tensorflow-2.1-gpu-py36]

## Available Amazon SageMaker Kernels

The following Amazon SageMaker kernels are available in SageMaker Studio. The name in parentheses is the SageMaker image hosting the kernel.

Data Science is a [Conda](#) image with the most commonly used Python packages and libraries, such as NumPy and scikit-learn.

- Python 3 (Data Science)
- PySpark (SparkMagic)
- Python 3 (Base Python)
- Python 3 (MXNet 1.6 Python 3.6 CPU Optimized)
- Python 3 (MXNet 1.6 Python 3.6 GPU Optimized)
- Python 3 (MXNet 1.8 Python 3.7 CPU Optimized)
- Python 3 (MXNet 1.8 Python 3.7 GPU Optimized)
- Python 3 (PyTorch 1.4 Python 3.6 CPU Optimized)
- Python 3 (PyTorch 1.4 Python 3.6 GPU Optimized)
- Python 3 (PyTorch 1.6 Python 3.6 CPU Optimized)
- Python 3 (PyTorch 1.6 Python 3.6 GPU Optimized)
- Python 3 (SageMaker JumpStart Data Science 1.0)

- Python 3 (SageMaker JumpStart MXNet 1.0)
- Python 3 (SageMaker JumpStart PyTorch 1.0)
- Python 3 (SageMaker JumpStart TensorFlow 1.0)
- Python 3 (TensorFlow 1.15 Python 3.6 CPU Optimized)
- Python 3 (TensorFlow 1.15 Python 3.6 GPU Optimized)
- Python 3 (TensorFlow 1.15 Python 3.7 CPU Optimized)
- Python 3 (TensorFlow 1.15 Python 3.7 GPU Optimized)
- Python 3 (TensorFlow 2.1 Python 3.6 CPU Optimized)
- Python 3 (TensorFlow 2.1 Python 3.6 GPU Optimized)
- Python 3 (TensorFlow 2.3 Python 3.7 CPU Optimized)
- Python 3 (TensorFlow 2.3 Python 3.7 GPU Optimized)

## Bring your own SageMaker image

A SageMaker image is a file that identifies the kernels, language packages, and other dependencies required to run a Jupyter notebook in Amazon SageMaker Studio. These images are used to create an environment that you then run the Jupyter notebooks from. Amazon SageMaker provides many built-in images for you to use. If you need different functionality, you can bring your own custom images to Studio. For the list of built-in images, see [Available Amazon SageMaker Images \(p. 94\)](#).

A SageMaker *image* is a holder for a set of SageMaker *image versions*. An image version represents a container image that is compatible with SageMaker Studio and stored in an Amazon Elastic Container Registry (ECR) repository. Each image version is immutable.

To make a custom SageMaker image available to all users within a domain, you attach the image to the domain. To make an image available to a single user, you attach the image to the user's profile. When you attach an image, SageMaker uses the latest image version by default. You can also attach a specific image version. After you attach the version, you can choose the version from the SageMaker Launcher or the image selector when you launch a notebook.

You can create images and image versions, and attach image versions to your domain, using the SageMaker Studio control panel, the [Amazon SDK for Python \(Boto3\)](#), and the [Amazon Command Line Interface \(Amazon CLI\)](#). You can also create images and image versions using the SageMaker console, even if you haven't onboarded to Studio.

SageMaker provides sample Dockerfiles to use as a starting point for your custom SageMaker images in the [SageMaker Studio Custom Image Samples](#) repository. These include Dockerfiles for the following images:

- Julia
- R
- Scala
- Tensorflow 2

The following topics explain how to bring your own image using the SageMaker console and then launch the image in SageMaker Studio. A more comprehensive tutorial additionally shows you how to build a custom container image from a supplied R Dockerfile. For a similar blog article, see [Bringing your own R environment to Amazon SageMaker Studio](#).

### Topics

- [Create a custom SageMaker image \(Console\) \(p. 96\)](#)
- [Attach a custom SageMaker image \(Control Panel\) \(p. 97\)](#)

- [Launch a custom SageMaker image in SageMaker Studio \(p. 98\)](#)
- [Bring your own custom SageMaker image tutorial \(p. 101\)](#)
- [Custom SageMaker image specifications \(p. 109\)](#)

## Create a custom SageMaker image (Console)

This topic describes how you can create a custom SageMaker image using the SageMaker console. You can also create the image using the SageMaker Studio control panel. The steps are the same. For information on using the Studio control panel, see [Attach a custom SageMaker image \(Control Panel\) \(p. 97\)](#).

When you create an image, SageMaker also creates an initial image version. The image version represents a container image in [Amazon Elastic Container Registry \(ECR\)](#). The container image must satisfy the requirements to be used in Amazon SageMaker Studio. For more information, see [Custom SageMaker image specifications \(p. 109\)](#).

For information on testing your image locally and resolving common issues, see the [SageMaker Studio Custom Image Samples repo](#).

### To create an image

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the left navigation pane, choose **Images**.
3. On the **Custom images** page, choose **Create image**.
4. For **Image source**, enter the registry path to the container image in Amazon ECR. The path is in the following format:

`acct-id.dkr.ecr.region.amazonaws.com/repo-name[:tag] or [@digest]`

5. Choose **Next**.
6. Under **Image properties**, enter the following:
  - Image name – The name must be unique to your account in the current Amazon Region.
  - (Optional) Display name – The name displayed in the Studio user interface. When not provided, `Image name` is displayed.
  - (Optional) Description – A description of the image.
  - IAM role – The role must have the [AmazonSageMakerFullAccess](#) policy attached. Use the dropdown menu to choose one of the following options:
    - Create a new role – Specify any additional Amazon Simple Storage Service (Amazon S3) buckets that you want users of your notebooks to have access to. If you don't want to allow access to additional buckets, choose **None**.

SageMaker attaches the [AmazonSageMakerFullAccess](#) policy to the role. The role allows users of your notebooks access to the S3 buckets listed next to the checkmarks.

- Enter a custom IAM role ARN – Enter the Amazon Resource Name (ARN) of your IAM role.
- Use existing role – Choose one of your existing roles from the list.
- (Optional) Image tags – Choose **Add new tag**. You can add up to 50 tags. Tags are searchable using the Studio user interface, the SageMaker console, or the SageMaker Search API.

7. Choose **Submit**.

The new image is displayed in the **Custom images** list and briefly highlighted. After the image has been successfully created, you can choose the image name to view its properties or choose **Create version** to create another version.

### To create another image version

1. Choose **Create version** on the same row as the image.
2. For **Image source**, enter the registry path to the ECR container image. The container image shouldn't be the same image as used in a previous version of the SageMaker image.

To use the custom image in Studio, you must attach it to your domain. For more information, see [Attach a custom SageMaker image \(Control Panel\) \(p. 97\)](#).

## Attach a custom SageMaker image (Control Panel)

To use a custom SageMaker image, you must attach a version of the image to your domain. When you attach an image version, it appears in the SageMaker Studio Launcher and is available in the **Select image** dropdown list, which users use to launch an activity or change the image used by a notebook.

There is a limit to the number of image versions that can be attached at any given time. After you reach the limit, you must detach a version in order to attach another version of the image.

### Attach an existing image version to your domain

This topic describes how you can attach an existing custom SageMaker image version to your domain using the SageMaker Studio control panel. You can also create a custom SageMaker image and image version, and then attach that version to your domain.

The steps to create an image and image version are the same whether you use the Studio control panel or the SageMaker console. For the procedure to create an image and image version, see [Create a custom SageMaker image \(Console\) \(p. 96\)](#).

#### To attach an existing image

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the left navigation pane, choose **Amazon SageMaker Studio**.
3. On the **SageMaker Studio Control Panel**, under **Custom images attached to domain**, choose **Attach image**.
4. For **Image source**, choose **Existing image**.
5. Choose an existing image from the list.
6. Choose a version of the image from the list.
7. Choose **Next**.
8. Choose the IAM role. For more information, see [Create a custom SageMaker image \(Console\) \(p. 96\)](#).
9. Choose **Next**.
10. Under **Studio configuration**, enter or change the following settings. For information on how to get the kernel information from the image, see **DEVELOPMENT** in the SageMaker Studio Custom Image Samples repository.
  - EFS mount path – The path within the image to mount the user's Amazon Elastic File System (EFS) home directory.
  - Kernel:
    - For **Kernel name**, enter the name of an existing kernel in the image.
    - (Optional) For **Kernel display name**, enter the display name for the kernel.
    - Choose **Add kernel**.
  - (Optional) Configuration tags – Choose **Add new tag** and then add a configuration tag.

**Note**

For more information, see the **Kernel discovery** and **User data** sections of [Custom SageMaker image specifications \(p. 109\)](#).

11. Choose **Submit**

Wait for the image version to be attached to the domain. When attached, the version is displayed in the **Custom images** list and briefly highlighted.

## Detach a custom SageMaker image

When you detach an image from a domain, all versions of the image are detached. When an image is detached, all users of the domain lose access to the image versions.

A running notebook that has a kernel session on an image version when the version is detached, continues to run. When the notebook is stopped or the kernel is shut down the image version becomes unavailable.

### To detach an image

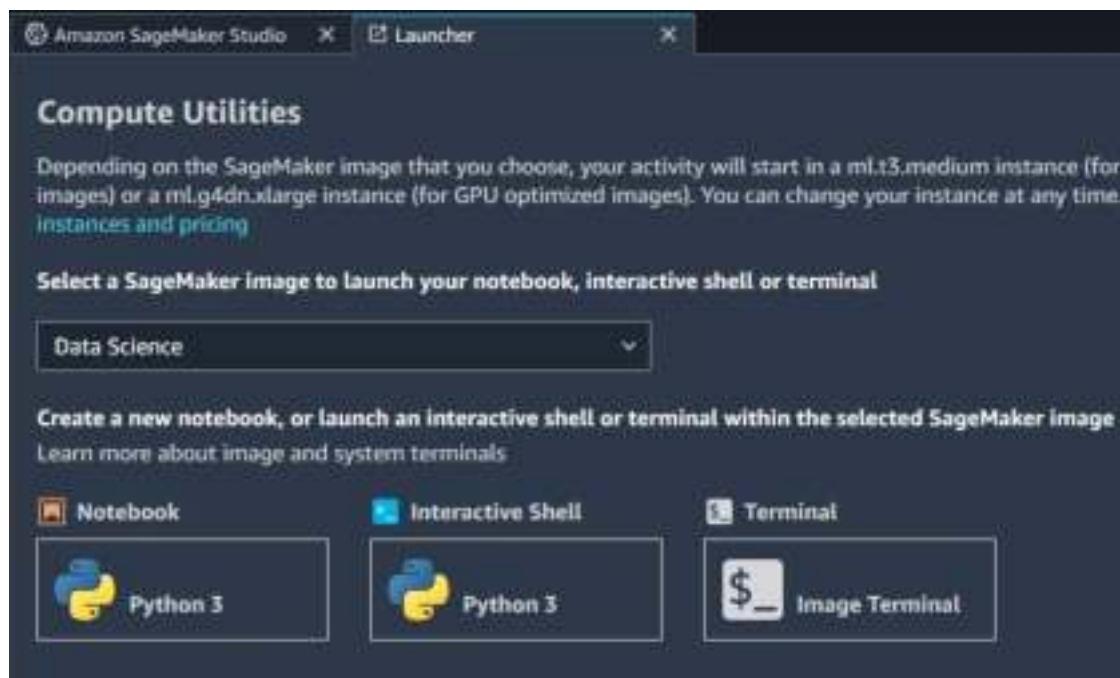
1. In the SageMaker Studio control panel, under **Custom images attached to domain**, choose the image and then choose **Detach**.
2. (Optional) To delete the image and all versions from SageMaker, select **Also delete the selected images** .... This does not delete the associated container images from Amazon ECR.
3. Choose **Detach**.

## Launch a custom SageMaker image in SageMaker Studio

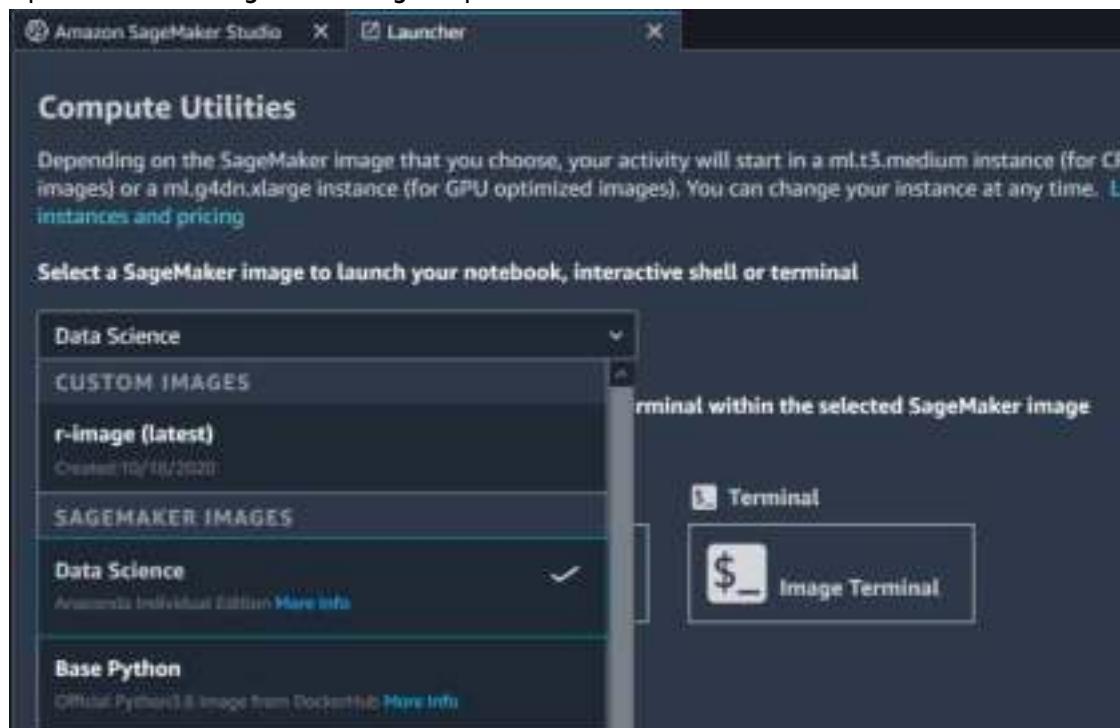
After you create your custom SageMaker image and attach it to your domain, the image appears in the image selector dialog box of the SageMaker Studio Launcher, and the kernel appears in the kernel selector dialog box.

### To launch and select your custom image

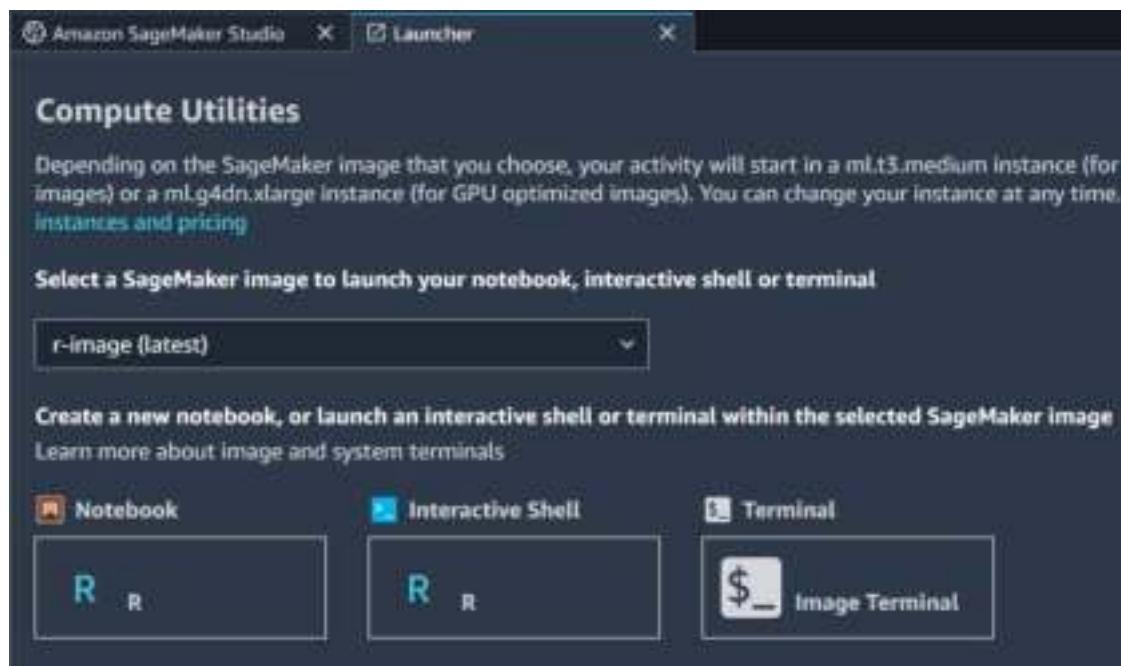
1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Use the keyboard shortcut **Ctrl + Shift + I** to open Studio Launcher.



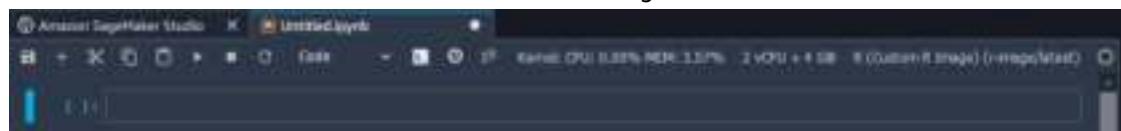
3. Open the Select a SageMaker image dropdown menu.



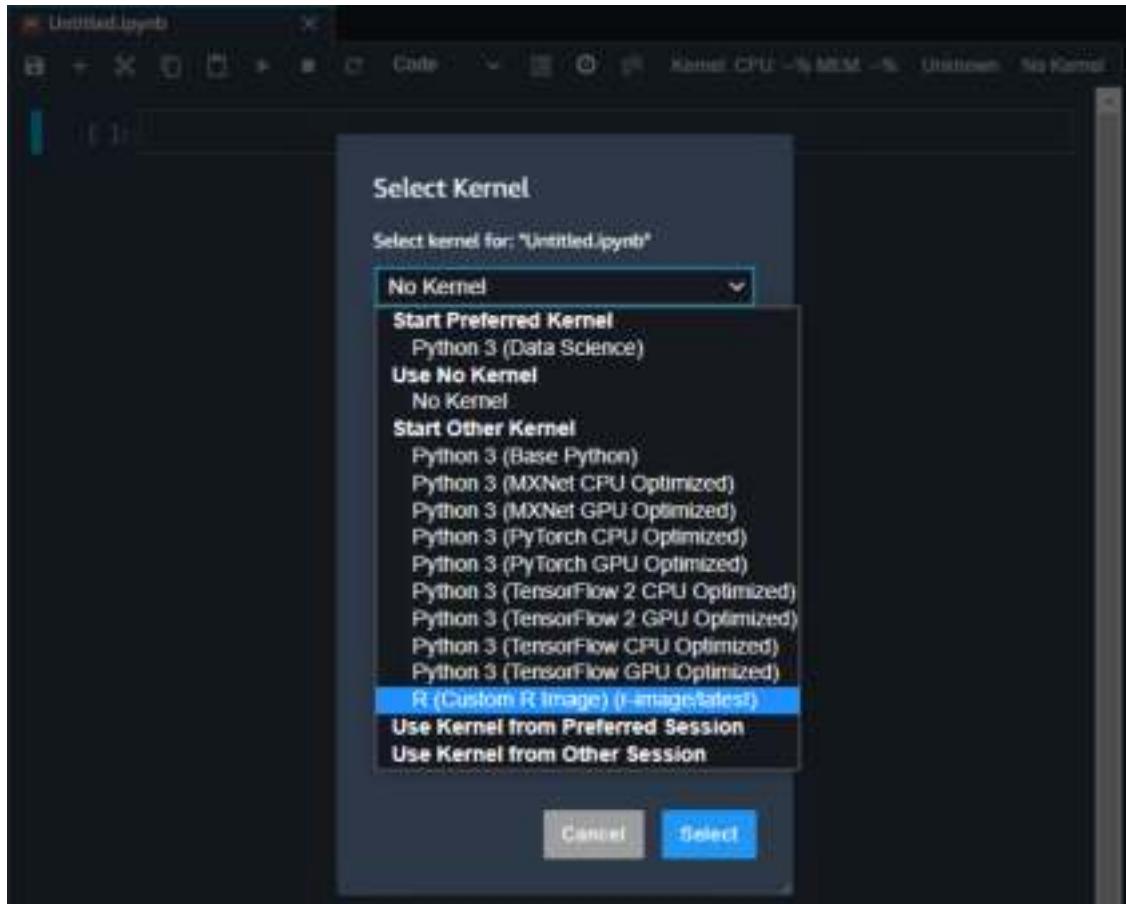
4. Choose your custom image.



5. Launch a notebook or interactive shell in the custom image.



6. In an open notebook, you can switch to the custom kernel by choosing a different kernel in the **Select Kernel** dialog box.



**Note**

If you encounter an error when launching the image, check your Amazon CloudWatch logs. The name of the log group is /aws/sagemaker/studio. The name of the log stream is \$domainID/\$userProfileName/KernelGateway/\$appName.

## Bring your own custom SageMaker image tutorial

In this tutorial, you create a custom SageMaker image and attach a version of the image to your domain for use in Amazon SageMaker Studio. The image version contains a selection of R packages, along with the Amazon SDK for Python (Boto3) and the [Amazon SageMaker Python SDK](#). After you complete this tutorial, you can select the version in Studio and use R to access the SDKs using the RStudio reticulate package. For more information on the reticulate package, see [R Interface to Python](#). For a blog article similar to this tutorial, see [Bringing your own R environment to Amazon SageMaker Studio](#).

Two methods are presented to attach the image version to your domain. In the first method, you create a new domain with the version attached. This method is simpler but you need to specify the Amazon Virtual Private Cloud (VPC) information and execution role that's required to create the domain.

If you have onboarded to Studio, you can use the second method to attach the image version to your current domain. In this case, you don't need to specify the VPC information and execution role. After you attach the version, you must delete all the apps in your domain and reopen Studio.

You can't run this tutorial from Studio because you can't create or update a domain from within Studio.

## Prerequisites

- The Docker application. For information about setting up Docker, see [Orientation and setup](#).
- A local copy of the Dockerfile for creating a Studio compatible [R image](#) from the SageMaker Studio [custom image samples](#) repository.

### Note

Building the R image from the Dockerfile installs dependencies that may be licensed under copyleft licenses such as GPLv3. You should review the license terms and make sure they are acceptable for your use case before proceeding and building this image.

- Permissions to access the Amazon Elastic Container Registry (Amazon ECR) service. For more information, see [Amazon ECR Managed Policies](#).
- An Amazon Identity and Access Management execution role that has the [AmazonSageMakerFullAccess](#) policy attached. If you have onboarded to Amazon SageMaker Studio, you can get the role from the [Studio Summary](#) section of the SageMaker Studio control panel.

## Links

- [Amazon Elastic Container Registry API Reference](#)
- [Docker Engine API](#)
- [Dockerfile reference](#)

## Topics

- [Add a Studio-compatible container image to Amazon ECR \(p. 102\)](#)
- [Create a SageMaker image from the ECR container image \(p. 103\)](#)
- [Attach the SageMaker image to a new domain \(p. 105\)](#)
- [Attach the SageMaker image to your current domain \(p. 106\)](#)
- [View the attached image in the Studio control panel \(p. 108\)](#)
- [Clean up resources \(p. 108\)](#)

## Add a Studio-compatible container image to Amazon ECR

You perform the following steps to add a container image to Amazon ECR:

- Create an Amazon ECR repository.
- Authenticate to Amazon ECR.
- Build a Studio-compatible container image.
- Push the image to the Amazon ECR repository.

### Note

The Amazon ECR repository must be in the same Amazon Region as SageMaker Studio.

### To build and add a container image to Amazon ECR

1. Create an Amazon ECR repository using the Amazon CLI. To create the repository using the Amazon ECR console, see [Creating a repository](#).

```
aws ecr create-repository \
    --repository-name smstudio-custom \
    --image-scanning-configuration scanOnPush=true
```

Response:

```
{  
    "repository": {  
        "repositoryArn": "arn:aws:ecr:us-east-2:acct-id:repository/smstudio-custom",  
        "registryId": "acct-id",  
        "repositoryName": "smstudio-custom",  
        "repositoryUri": "acct-id.dkr.ecr.us-east-2.amazonaws.com/smstudio-custom",  
        ...  
    }  
}
```

2. Install the SageMaker Studio image build CLI by following the steps in [SageMaker Docker Build](#). This CLI enables you to build a Dockerfile using Amazon CodeBuild.
3. Build the R image Dockerfile using the Studio image build CLI. The period (.) specifies that the Dockerfile should be in the context of the build command. This command builds the image, creates an ECR repo, and uploads the built image to the ECR repo. It then outputs the image URI.

```
sm-docker build . -t smstudio-r -t <acct-id>.dkr.ecr.<region>.amazonaws.com/smstudio-  
custom:r
```

Response:

```
Image URI: <acct-id>.dkr.ecr.<region>.amazonaws.com/<image_name>
```

## Create a SageMaker image from the ECR container image

You perform the following steps to create a SageMaker image from the container image:

- Create an Image.
- Create an ImageVersion.
- Create a configuration file.
- Create an AppImageConfig.

### To create the SageMaker image entities

1. Create a SageMaker image.

```
aws sagemaker create-image \  
    --image-name r-image \  
    --role-arn arn:aws:iam::<acct-id>:role/service-role/<execution-role>
```

Response:

```
{  
    "ImageArn": "arn:aws:sagemaker:us-east-2:acct-id:image/r-image"  
}
```

2. Create a SageMaker image version from the container image.

```
aws sagemaker create-image-version \  
    --image-name r-image \  
    --base-image <acct-id>.dkr.ecr.<region>.amazonaws.com/smstudio-custom:r
```

Response:

```
{  
    "ImageVersionArn": "arn:aws:sagemaker:us-east-2:acct-id:image-version/r-image/1"  
}
```

3. Check that the image version was successfully created.

```
aws sagemaker describe-image-version \  
    --image-name r-image \  
    --version 1
```

Response:

```
{  
    "ImageVersionArn": "arn:aws:sagemaker:us-east-2:acct-id:image-version/r-image/1",  
    "ImageVersionStatus": "CREATED"  
}
```

**Note**

If the response is "ImageVersionStatus": "CREATED\_FAILED", the response also includes the failure reason. A permissions issue is a common cause of failure. You also can check your Amazon CloudWatch logs. The name of the log group is /aws/sagemaker/studio. The name of the log stream is \$domainID/\$userProfileName/KernelGateway/\$appName.

4. Create a configuration file, named `app-image-config-input.json`. The `Name` value of `KernelSpecs` must match the name of the `kernelSpec` available in the `Image` associated with this `AppImageConfig`. This value is case sensitive. You can find the available `kernelSpecs` in an image by running `jupyter-kernelspec list` from a shell inside the container. For information on testing your image locally before using it in Studio, see [DEVELOPMENT](#) in the SageMaker Studio Custom Image Samples repository.

```
{  
    "AppImageConfigName": "r-image-config",  
    "KernelGatewayImageConfig": {  
        "KernelSpecs": [  
            {  
                "Name": "ir",  
                "DisplayName": "R (Custom R Image)"  
            }  
        ],  
        "FileSystemConfig": {  
            "MountPath": "/home/sagemaker-user",  
            "DefaultUid": 1000,  
            "DefaultGid": 100  
        }  
    }  
}
```

5. Create the `AppImageConfig` using the file created in the previous step.

```
aws sagemaker create-app-image-config \  
    --cli-input-json file://app-image-config-input.json
```

Response:

```
{
```

```
    "AppImageConfigArn": "arn:aws:sagemaker:us-east-2:acct-id:app-image-config/r-image-  
config"  
}
```

## Attach the SageMaker image to a new domain

To use this method, you need to specify an execution role that has the [AmazonSageMakerFullAccess](#) policy attached.

**Note**

You can have only one domain. If you have onboarded to SageMaker Studio, you must delete your current domain before you can use this method. For more information, see [Delete an Amazon SageMaker Studio Domain \(p. 39\)](#).

You perform the following steps to create the domain and attach the custom SageMaker image:

- Get your default VPC ID and subnet IDs.
- Create the configuration file for the domain, which specifies the image.
- Create the domain with the configuration file.

### To add the custom SageMaker image to your domain

1. Get your default VPC ID.

```
aws ec2 describe-vpcs \  
  --filters Name=isDefault,Values=true \  
  --query "Vpcs[0].VpcId" --output text
```

Response:

```
vpc-xxxxxxxx
```

2. Get your default subnet IDs using the VPC ID from the previous step.

```
aws ec2 describe-subnets \  
  --filters Name=vpc-id,Values=<vpc-id> \  
  --query "Subnets[*].SubnetId" --output json
```

Response:

```
[  
  "subnet-b55171dd",  
  "subnet-8a5f99c6",  
  "subnet-e88d1392"  
]
```

3. Create a configuration file named `create-domain-input.json`. Insert the VPC ID, subnet IDs, `ImageName`, and `AppImageConfigName` from the previous steps. Because `ImageVersionNumber` isn't specified, the latest version of the image is used, which is the only version in this case.

```
{  
  "DomainName": "domain-with-custom-r-image",  
  "VpcId": "<vpc-id>",  
  "SubnetIds": [  
    "<subnet-ids>"  
  ],
```

```
"DefaultUserSettings": {  
    "ExecutionRole": "<execution-role>",  
    "KernelGatewayAppSettings": {  
        "CustomImages": [  
            {  
                "ImageName": "r-image",  
                "AppImageConfigName": "r-image-config"  
            }  
        ]  
    },  
    "AuthMode": "IAM"  
}
```

4. Create the domain with the attached custom SageMaker image.

```
aws sagemaker create-domain \  
--cli-input-json file://create-domain-input.json
```

Response:

```
{  
    "DomainArn": "arn:aws:sagemaker:us-east-2:acct-id:domain/d-xxxxxxxxxxxxxx",  
    "Url": "https://d-xxxxxxxxxxxxx.studio.us-east-2.sagemaker.aws/..."  
}
```

## Attach the SageMaker image to your current domain

This method presumes you've already onboarded to Amazon SageMaker Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).

### Note

You must delete all the apps in your domain before you update the domain with the new image version. For information about deleting the apps, see [Delete an Amazon SageMaker Studio Domain \(p. 39\)](#).

You perform the following steps to add the SageMaker image to your current domain.

- Get your `DomainID` from SageMaker Studio.
- Use the `DomainID` to get the `DefaultUserSettings` for the domain.
- Add the `ImageName` and `AppImageConfig` as a `CustomImage` to the `DefaultUserSettings`.
- Update your domain to include the custom image.

### To add the custom SageMaker image to your domain

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. From the top left of the navigation pane, choose **Amazon SageMaker Studio**.
3. From the Studio Control Panel, under **Studio Summary**, find the **Studio ID**, which is also your `DomainId`. The ID is in the following format: `d-xxxxxxxxxxxxxx`.

The screenshot shows the SageMaker Studio Control Panel. On the left, there's a sidebar with links like Dashboard, Search, Images, Ground Truth, Labeling jobs, Labeling datasets, Labeling workforces, Notebook, Notebook instances, Lifecycle configurations, and Git repositories. The main area has a header "Amazon SageMaker > SageMaker Studio > Control Panel". Below it, a search bar says "Choose your user name, then choose Open Studio". A table lists users: "demo-user" with status "Oct 16, 2020 16:34 UTC". Under "Studio Summary", it shows "Status: Ready" and "Studio ID: d-xxxxxxxxxxxxxx" (the last part is highlighted with a red box).

4. Use the domain ID to get the description of the domain.

```
aws sagemaker describe-domain \
--domain-id <d-xxxxxxxxxxxxxx>
```

Response:

```
{
    "DomainId": "d-xxxxxxxxxxxxxx",
    "DefaultUserSettings": {
        "KernelGatewayAppSettings": {
            "CustomImages": [
                ],
                ...
            }
        }
}
```

5. Save the default user settings section of the response to a file named `default-user-settings.json`.
6. Insert the `ImageName` and `AppImageConfigName` from the previous steps as a custom image. Because `ImageVersionNumber` isn't specified, the latest version of the image is used, which is the only version in this case.

```
{
    "DefaultUserSettings": {
        "KernelGatewayAppSettings": {
            "CustomImages": [
                {
                    "ImageName": "string",
                    "AppImageConfigName": "string"
                }
            ],
            ...
        }
    }
}
```

```
        }
    }
```

7. Use the domain ID and default user settings file to update your domain.

```
aws sagemaker update-domain \
--domain-id <d-xxxxxxxxxxxx> \
--cli-input-json file://default-user-settings.json
```

Response:

```
{
    "DomainArn": "arn:aws:sagemaker:us-east-2:acct-id:domain/d-xxxxxxxxxxxx"
}
```

## View the attached image in the Studio control panel

After you create the custom SageMaker image and attach it to your domain, the image appears in the custom images list in the SageMaker Studio control panel.

The screenshot shows a table titled 'Custom images attached to domain'. The table has columns: Name, DisplayName, Latest attached version, Created, and Actions. There is one row visible with the following data: Name - 'y-image', DisplayName - 'y-image', Latest attached version - '1', Created - 'Oct 18, 2020 19:35 UTC', and Actions - 'Attach version'. At the top right of the table area, there are buttons for 'Detach' and 'Attach image'.

For information about how to launch the image in Amazon SageMaker Studio, see [Launch a custom SageMaker image in SageMaker Studio \(p. 98\)](#).

## Clean up resources

You perform the following steps to clean up the resources you created in the previous sections:

- Detach the image and image versions from your domain.
- Delete the image, image version, and app image config.
- Delete the container image and repository from Amazon ECR. For more information, see [Deleting a repository](#).

### To clean up resources

1. Detach the image and image versions from your domain by passing an empty custom image list to the domain. Open the `default-user-settings.json` file you created in [Attach the image to a current domain \(p. 106\)](#).
2. Delete the custom images and then save the file.

```
"DefaultUserSettings": {
    "KernelGatewayAppSettings": {
        "CustomImages": [
            ],
        ...
    },
    ...
}
```

```
}
```

3. Use the domain ID and default user settings file to update your domain.

```
aws sagemaker update-domain \
--domain-id <d-xxxxxxxxxxxx> \
--cli-input-json file://default-user-settings.json
```

Response:

```
{  
    "DomainArn": "arn:aws:sagemaker:us-east-2:acct-id:domain/d-xxxxxxxxxxxx"  
}
```

4. Delete the app image config.

```
aws sagemaker delete-app-image-config \
--app-image-config-name r-image-config
```

5. Delete the SageMaker image, which also deletes all image versions. The container images in ECR that are represented by the image versions are not deleted.

```
aws sagemaker delete-image \
--image-name r-image
```

## Custom SageMaker image specifications

The following specifications apply to the container image that is represented by a SageMaker image version.

### Running the image

ENTRYPOINT and CMD instructions are overridden to enable the image to run as a KernelGateway app.

Port 8888 in the image is reserved for running the KernelGateway web server.

### Stopping the image

The DeleteApp API issues the equivalent of a docker stop command. Other processes in the container won't get the SIGKILL/SIGTERM signals.

### Kernel discovery

SageMaker recognizes kernels as defined by Jupyter [kernel specs](#).

You can specify a list of kernels to display before running the image. If not specified, python3 is displayed. Use the [DescribeAppImageConfig](#) API to view the list of kernels.

Conda environments are recognized as kernel specs by default.

### File system

The /opt/.sagemakerinternal and /opt/ml directories are reserved. Any data in these directories might not be visible at runtime.

### User data

Each user in a Studio domain gets a user directory on a shared Amazon Elastic File System volume in the image. The location of the current user's directory on the Amazon EFS volume is configurable. By default, the location of the directory is /home/sagemaker-user.

SageMaker configures POSIX UID/GID mappings between the image and the host. This defaults to mapping the root user's UID/GID (0/0) to the UID/GID on the host.

You can specify these values using the [CreateAppImageConfig](#) API.

#### GID/UID Limits

SageMaker Studio only supports UID and GID values in the range between 0 and 65535. This limit applies to files in every layer of the image.

#### Metadata

A metadata file is located at `/opt/ml/metadata/resource-metadata.json`. No additional environment variables are added to the variables defined in the image. For more information, see [Get App Metadata \(p. 84\)](#).

#### GPU

On a GPU instance, the image is run with the `--gpus` option. Only the CUDA toolkit should be included in the image not the NVIDIA drivers. For more information, see [NVIDIA User Guide](#).

#### Metrics and Logging

Logs from the KernelGateway process are sent to Amazon CloudWatch in the customer's account. The name of the log group is `/aws/sagemaker/studio`. The name of the log stream is `$domainID/$userProfileName/KernelGateway/$appName`.

#### Image size

Limited to 11 GB.

## Set Up a Connection to an Amazon EMR Cluster

Amazon EMR is a big data platform for processing vast amounts of data. The central component of Amazon EMR is the cluster. A cluster is a collection of Amazon EC2 instances. Apache Spark is a distributed processing framework that runs on Amazon EMR. For more information, see [What Is Amazon EMR?](#) and [Apache Spark](#).

Amazon SageMaker Studio comes with a SageMaker SparkMagic image that contains a PySpark kernel. The SparkMagic image also contains an Amazon CLI utility, `sm-sparkmagic`, that you can use to create the configuration files required for the PySpark kernel to connect to the Amazon EMR cluster. After creating the configuration files, the utility displays the steps required to finish the setup.

For added security, you can specify that the connection to the EMR cluster uses Kerberos authentication. For more information, see [Use Kerberos Authentication](#).

#### Prerequisites

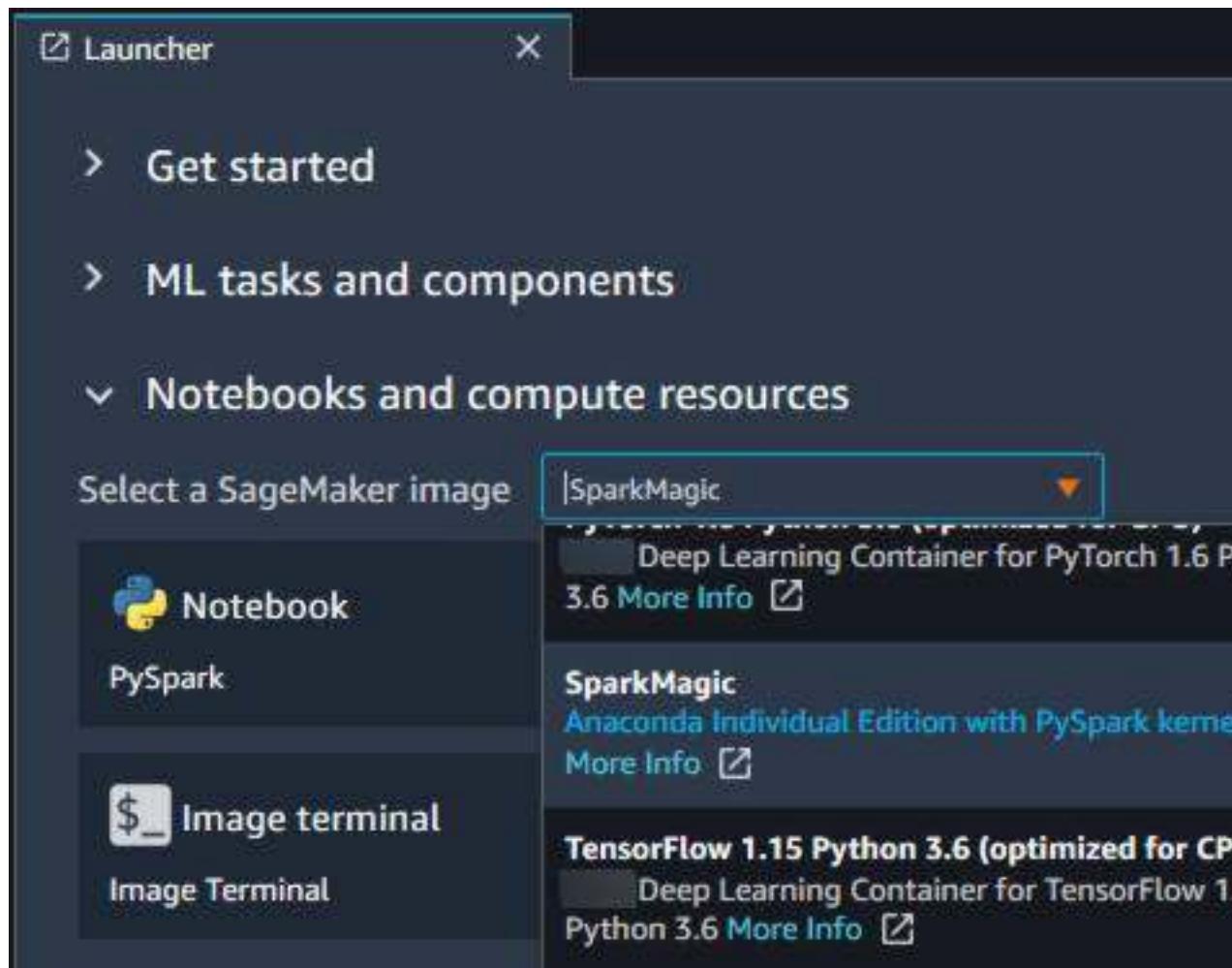
- Access to SageMaker Studio that's set up to use Amazon VPC mode. To connect to Amazon EMR, Studio must be configured as Amazon VPC only mode. For more information, see [Connect SageMaker Studio Notebooks to Resources in a VPC \(p. 2491\)](#).
- An Amazon EMR cluster in the same VPC as Studio or in a VPC that's connected to the same VPC as Studio. This cluster must have Spark and Livy installed.
- The security group used for Amazon SageMaker Studio and the Amazon EMR security group must allow access to and from each other.
- Your Amazon EMR security group must open port 8998, so Amazon SageMaker Studio can communicate with the Spark cluster via Livy. For more information on setting up the security group, see [Build SageMaker notebooks backed by Spark in Amazon EMR](#).

- If you use the `sm-sparkmagic` utility, the IAM execution role associated with your Studio user profile must contain the following extra permissions. To find the execution role, choose your user name in the SageMaker Studio Control Panel.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "elasticmapreduce:DescribeCluster",  
                "elasticmapreduce:DescribeSecurityConfiguration",  
                "elasticmapreduce>ListInstances"  
            ],  
            "Resource": [  
                "arn:aws:elasticmapreduce:*::cluster/*"  
            ]  
        }  
    ]  
}
```

### To set up a connection to an EMR cluster

1. Open SageMaker Studio.
2. In the upper-left corner of Studio, choose **Amazon SageMaker Studio** to open Studio Launcher.
3. On the **Launcher** page, choose **Notebooks and compute resources**.
4. For **Select a SageMaker image**, choose the **SparkMagic** image.



5. Choose **Notebook** to create a Studio notebook in the SparkMagic image.
6. Run the following code in a notebook cell to create the configuration files used to connect to the EMR cluster. `%%local` ensures that the code runs in the local image instead of on Spark.
  - If the EMR cluster is not configured for Kerberos authentication, run the following command:

```
%%local
! sm-sparkmagic connect --cluster-id "cluster-id"
```

The output should be similar to the following:

```
Successfully read emr cluster(cluster-id) details
SparkMagic config file location: /etc/sparkmagic/config.json
```

- If the EMR cluster is configured for Kerberos authentication, run the following command:

```
! sm-sparkmagic connect --cluster-id "cluster-id" --user-name "user-name"
```

The output should be similar to the following:

```
Successfully read emr cluster(cluster-id) details
```

```
SparkMagic config file location: /etc/sparkmagic/config.json
Kerberos configuration file location: /etc/krb5.conf
```

7. To complete the setup, do one of the following:
  - For EMR clusters that are not configured for Kerberos authentication, go to step 8.
  - For EMR clusters that are configured for Kerberos authentication, do the following:
    1. In the notebook toolbar, choose the **Launch terminal** icon () to open a terminal in the same SparkMagic image as the notebook.
    2. Run the following command in the terminal to get the Kerberos ticket:
8. In the notebook toolbar, choose the **Restart kernel** icon () to complete the setup.
9. To verify that the connection was set up correctly, run the following command in a notebook cell:

```
%%info
```

The output should be similar to the following:

```
Current session configs:{'driverMemory': '1000M', 'executorCores': 2, 'kind': 'pyspark'}
No active sessions.
```

### For more information

- [Perform interactive data processing using Spark in Amazon SageMaker Studio Notebooks](#).

## Perform Common Tasks in Amazon SageMaker Studio

The following sections describe how to perform common tasks in Amazon SageMaker Studio. For an overview of the Studio interface, see [Amazon SageMaker Studio UI Overview \(p. 69\)](#).

### Topics

- [Upload Files to SageMaker Studio \(p. 114\)](#)
- [Clone a Git Repository in SageMaker Studio \(p. 114\)](#)
- [Stop a Training Job in SageMaker Studio \(p. 114\)](#)
- [Use TensorBoard in Amazon SageMaker Studio \(p. 115\)](#)
- [Manage Your EFS Storage Volume in SageMaker Studio \(p. 116\)](#)
- [Provide Feedback on SageMaker Studio \(p. 117\)](#)
- [Update SageMaker Studio and Studio Apps \(p. 117\)](#)

## Upload Files to SageMaker Studio

When you onboard to Amazon SageMaker Studio, a home directory is created for you in the Amazon Elastic File System (Amazon EFS) volume that was created for your team. Studio can open only files that have been uploaded to your directory. The Studio file browser maps to your home directory.

### To upload files to your home directory

1. In the left sidebar, choose the **File Browser** icon ().
2. In the file browser, choose the **Upload Files** icon ().
3. Select the files you want to upload and then choose **Open**.
4. Double-click a file to open the file in a new tab in Studio.

## Clone a Git Repository in SageMaker Studio

Amazon SageMaker Studio can connect only to a local repository. In this example, you clone the [aws/amazon-sagemaker-examples](#) repository (repo).

### To clone the repo

1. In the left sidebar, choose the **File Browser** icon ().
2. Choose the root folder or the folder you want to clone the repo into.
3. In the left sidebar, choose the **Git** icon ().
4. Choose **Clone a Repository**.
5. Enter the URI for the SageMaker examples repo <https://github.com/aws/amazon-sagemaker-examples.git>.
6. Choose **CLONE**.
7. If the repo requires credentials, you are prompted to enter your username and password.
8. Wait for the download to finish. After the repo has been cloned, the **File Browser** opens to display the cloned repo.
9. Double click the repo to open it.
10. Choose the **Git** icon to view the Git user interface which now tracks the examples repo.
11. To track a different repo, open the repo in the file browser and then choose the **Git** icon.

## Stop a Training Job in SageMaker Studio

You can stop a training job with the Amazon SageMaker Studio UI. When you stop a training job, its status changes to **Stopping** at which time billing ceases. An algorithm can delay termination in order to save model artifacts after which the job status changes to **Stopped**. For more information, see the [stop\\_training\\_job](#) method in the Amazon SDK for Python (Boto3).

### To stop a training job

1. Follow the [View and Compare Experiments, Trials, and Trial Components \(p. 1558\)](#) procedure on this page until you open the **Describe Trial Component** tab.
2. At the upper-right side of the tab, choose **Stop training job**. The **Status** at the top left of the tab changes to **Stopped**.

3. To view the training time and billing time, choose **Amazon Settings**.

## Use TensorBoard in Amazon SageMaker Studio

The following doc outlines how to install and run TensorBoard in Amazon SageMaker Studio.

### Prerequisites

This tutorial requires an Amazon SageMaker Studio Domain.

### Set Up TensorBoardCallback

1. Launch Studio.
2. In the Amazon SageMaker Studio Launcher under Notebooks and compute resources, select the TensorFlow 2.3 Python 3.7(optimized for CPU) Studio Image.
3. Launch a notebook.
4. Import the required packages.

```
import os
import datetime
import tensorflow as tf
```

5. Create your Keras model.

```
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

def create_model():
    return tf.keras.models.Sequential([
        tf.keras.layers.Flatten(input_shape=(28, 28)),
        tf.keras.layers.Dense(512, activation='relu'),
        tf.keras.layers.Dropout(0.2),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
```

6. Create a directory for your TensorBoard logs

```
LOG_DIR = os.path.join(os.getcwd(), "logs/fit/" + datetime.datetime.now().strftime("%Y
%m%d-%H%M%S"))
```

7. Run training with TensorBoard.

```
model = create_model()
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=LOG_DIR,
                                                      histogram_freq=1)

model.fit(x=x_train,
          y=y_train,
          epochs=5,
          validation_data=(x_test, y_test),
```

```
 callbacks=[tensorboard_callback])
```

8. Generate the EFS path for the TensorBoard logs. You use this path to set up your logs from the terminal.

```
EFS_PATH_LOG_DIR = "/".join(LOG_DIR.strip("/").split('/')[1:-1])
print (EFS_PATH_LOG_DIR)
```

## Install TensorBoard

1. Click on the Amazon SageMaker Studio button on the top left corner of Studio to open the Amazon SageMaker Studio Launcher. This launcher must be opened from your root directory.
2. In the Launcher under Utilities and files, click System terminal.
3. From the terminal, run the following commands. Copy EFS\_PATH\_LOG\_DIR from the Jupyter notebook. You must run this from the /home/sagemaker-user root directory.

```
pip install tensorboard
tensorboard --logdir <EFS_PATH_LOG_DIR>
```

## Launch TensorBoard

1. To launch TensorBoard, copy your Studio URL and replace lab? with proxy/6006/ as follows. You must include the trailing / character.

```
https://<YOUR_URL>.studio.region.sagemaker.aws/jupyter/default/proxy/6006/
```

2. Navigate to the URL to examine your results.

## Manage Your EFS Storage Volume in SageMaker Studio

The first time a user on your team onboards to Amazon SageMaker Studio, Amazon SageMaker creates an Amazon Elastic File System (Amazon EFS) volume for the team. A home directory is created in the volume for each user who onboards to Studio as part of your team. Notebook files and data files are stored in these directories. Users don't have access to other team member's home directories.

### Important

Don't delete the Amazon EFS volume. If you delete it, the domain will no longer function and all of your users will lose their work.

### To find your Amazon EFS volume

1. From the SageMaker Studio Control Panel, under **Studio Summary**, find the **Studio ID**. The ID will be in the following format: d-xxxxxxxxxxxxxx.
2. Pass the **Studio ID**, as **DomainId**, to the **describe\_domain** method.
3. In the response from **describe\_domain**, note the value for the **HomeEfsFileSystemId** key. This is the Amazon EFS file system ID.
4. Open the [Amazon EFS console](#). Make sure the Amazon Region is the same Region that's used by Studio.
5. Under **File systems**, choose the file system ID from the previous step.

6. To verify that you've chosen the correct file system, select the **Tags** heading. The value corresponding to the `ManagedByAmazonSageMakerResource` key should match the Studio ID.

For information on how to access the Amazon EFS volume, see [Using file systems in Amazon EFS](#).

To delete the Amazon EFS volume, see [Deleting an Amazon EFS file system](#).

## Provide Feedback on SageMaker Studio

Amazon SageMaker takes your feedback seriously. We encourage you to provide feedback.

### To provide feedback

1. At the upper-right of SageMaker Studio, choose **Feedback**.
2. Choose a smiley emoji to let us know how satisfied you are with SageMaker Studio and add any feedback you'd care to share with us.
3. Decide whether to share your identity with us, then choose **Submit**.

## Update SageMaker Studio and Studio Apps

Amazon SageMaker Studio provides a notification icon ( ) in the upper-right corner. The number of unread notices is displayed in the icon. To read the notices, select the icon. Two types of notifications are provided:

- Upgrade – Displayed when Studio or one of the Studio apps have released a new version. To update, see [Update SageMaker Studio and Studio Apps \(p. 117\)](#).
- Information – Displayed for new features and other information.

To reset the notification icon, you must select the link in each notice.

The following procedures show you how to update Amazon SageMaker Studio and SageMaker Studio apps to their latest versions.

To learn how to update [Amazon SageMaker Data Wrangler](#), see [Update Studio Apps \(p. 118\)](#).

### Topics

- [Update SageMaker Studio \(p. 117\)](#)
- [Update Studio Apps \(p. 118\)](#)

## Update SageMaker Studio

To update Amazon SageMaker Studio to the latest release, you must shut down the JupyterServer app. Any unsaved notebook information is lost in the process. The user data in the Amazon EFS volume isn't impacted.

After the JupyterServer app is shut down, you must reopen Studio through the SageMaker Studio Control Panel which creates a new version of the JupyterServer app.

You can shut down the JupyterServer app from the Studio Control Panel or from within Studio.

Some of the services within Studio, like Data Wrangler, run on their own app. To update these services you must delete the app for that service. To learn more, see [Update Studio Apps \(p. 118\)](#).

**Note**

A JupyterServer app is associated with a single Studio user. When you update the app for one user it doesn't affect other users.

**To shut down the JupyterServer app from the Studio Control Panel**

1. Choose your user name.
2. Under **Apps**, in the row displaying **JupyterServer**, choose **Delete app**.
3. Choose **Yes, delete app**.
4. Type **delete** in the confirmation box.
5. Choose **Delete**.

**To shut down the JupyterServer app from inside Studio**

1. (Optional) View the current Studio version number.
  - a. Open the Studio Launcher. Choose **Amazon SageMaker Studio** in the top-left of Studio.
  - b. Open **Utilities and files**.
  - c. Choose **System terminal**.
  - d. Run the following command: `jupyter labextension list`  
The version is specified similar to `@amzn/sagemaker-ui v2.13.1`.
2. On the top menu, choose **File** then **Shut Down**.
3. Choose one of the following options:
  - **Shutdown Server** – Shuts down the JupyterServer app. Terminal sessions, kernel sessions, SageMaker images, and instances aren't shut down. These resources continue to accrue charges.
  - **Shutdown All** – Shuts down all apps, terminal sessions, kernel sessions, SageMaker images, and instances. These resources no longer accrue charges.
4. Close the window.

## Update Studio Apps

To update an Amazon SageMaker Studio app to the latest release, you must first shut down the corresponding KernelGateway app from the SageMaker Studio Control Panel. After the KernelGateway app is shut down, you must reopen it through SageMaker Studio by running a new kernel. The kernel automatically updates. Any unsaved notebook information is lost in the process. The user data in the Amazon EFS volume isn't impacted.

**Note**

A KernelGateway app is associated with a single Studio user. When you update the app for one user it doesn't effect other users.

**To shut down the KernelGateway app**

1. Choose your user name.
2. Under **Apps**, in the row displaying the **App name**, choose **Delete app**.

To update Data Wrangler, delete the app that starts with **sagemaker-data-wrang**.

3. Choose **Yes, delete app**.
4. Type **delete** in the confirmation box.
5. Choose **Delete**.

## Amazon SageMaker Studio Pricing

When the first member of your team onboards to Amazon SageMaker Studio, Amazon SageMaker creates an Amazon Elastic File System (Amazon EFS) volume for the team. In the SageMaker Studio Control Panel, when the Studio **Status** displays as **Ready**, the Amazon EFS volume has been created.

When this member, or any member of the team, opens Studio, a home directory is created in the volume for the member. A storage charge is incurred for this directory. Subsequently, additional storage charges are incurred for the notebooks and data files stored in the member's home directory. For pricing information on Amazon EFS, see [Amazon EFS Pricing](#).

Additional costs are incurred when other operations are run inside Studio, for example, creating an Amazon SageMaker Autopilot job, running a notebook, running training jobs, and hosting a model.

For information on the costs associated with using Studio notebooks, see [Usage Metering \(p. 92\)](#).

For information about billing along with pricing examples, see [Amazon SageMaker Pricing](#).

## Troubleshooting Amazon SageMaker Studio

The following are common errors that you might run into when using Amazon SageMaker Studio. Each error is followed by a solution to the error.

- **SageMaker Studio core functionalities are not available.**

If you get this error message when opening Studio, it might due to Python package version conflicts. This occurs if you used the following commands in a notebook or terminal to install Python packages that have version conflicts with SageMaker package dependencies.

```
!pip install
```

```
pip install --user
```

To resolve this issue, complete the following steps:

1. Uninstall recently installed Python packages. If you're not sure which package to uninstall, reach out using the feedback button on the lower left of the Amazon Web Services Management Console.
2. Restart Studio:
  - a. Shut down Studio from the **File** menu.
  - b. Wait for 1 minute.
  - c. Re-open Studio by refreshing the page or opening it from the Amazon Web Services Management Console.

The problem should be resolved if you have uninstalled the package which caused the conflict. To install packages without causing this issue again, use `%pip install` without the `--user` flag.

If the issue persists, create a new user profile and set up your environment with that user profile.

If these solutions don't fix the issue, reach out using the feedback button on the lower left of the Amazon Web Services Management Console.

# Use Amazon SageMaker Notebook Instances

An *Amazon SageMaker notebook instance* is a machine learning (ML) compute instance running the Jupyter Notebook App. SageMaker manages creating the instance and related resources. Use Jupyter notebooks in your notebook instance to prepare and process data, write code to train models, deploy models to SageMaker hosting, and test or validate your models.

SageMaker also provides sample notebooks that contain complete code walkthroughs. These walkthroughs show how to use SageMaker to perform common machine learning tasks. For more information, see [Example Notebooks \(p. 131\)](#).

This video shows you how to setup and use SageMaker notebook instances. (Length: 26:04)

This video is a deep dive on how to use SageMaker notebook instances. (Length: 16:44)

## Topics

- [Create a Notebook Instance \(p. 120\)](#)
- [Access Notebook Instances \(p. 122\)](#)
- [Update a Notebook Instance \(p. 123\)](#)
- [Customize a Notebook Instance Using a Lifecycle Configuration Script \(p. 124\)](#)
- [Example Notebooks \(p. 131\)](#)
- [Set the Notebook Kernel \(p. 132\)](#)
- [Associate Git Repositories with SageMaker Notebook Instances \(p. 133\)](#)
- [Notebook Instance Metadata \(p. 140\)](#)
- [Monitor Jupyter Logs in Amazon CloudWatch Logs \(p. 140\)](#)

## Create a Notebook Instance

An *Amazon SageMaker notebook instance* is a ML compute instance running the Jupyter Notebook App. SageMaker manages creating the instance and related resources. Use Jupyter notebooks in your notebook instance to prepare and process data, write code to train models, deploy models to SageMaker hosting, and test or validate your models.

To create a notebook instance, use either the SageMaker console or the [CreateNotebookInstance API](#).

The notebook instance type you choose depends on how you use your notebook instance. You want to ensure that your notebook instance is not bound by memory, CPU, or IO. If you plan to load a dataset into memory on the notebook instance for exploration or preprocessing, we recommend that you choose an instance type with enough RAM memory for your dataset. This would require an instance with at least 16 GB of memory (.xlarge or larger). If you plan to use the notebook for compute intensive preprocessing, we recommend you choose a compute-optimized instance such as a c4 or c5.

A best practice when using a SageMaker notebook is to use the notebook instance to orchestrate other Amazon services. For example, you can use the notebook instance to manage large dataset processing by making calls to Amazon Glue for ETL (extract, transform, and load) services or Amazon EMR for mapping and data reduction using Hadoop. You can use Amazon services as temporary forms of computation or storage for your data.

You can store and retrieve your training and test data using an Amazon S3 bucket. You can then use SageMaker to train and build your model, so the instance type of your notebook would have no bearing on the speed of your model training and testing.

After receiving the request, SageMaker does the following:

- **Creates a network interface**—If you choose the optional VPC configuration, SageMaker creates the network interface in your VPC. It uses the subnet ID that you provide in the request to determine which Availability Zone to create the subnet in. SageMaker associates the security group that you provide in the request with the subnet. For more information, see [Connect a Notebook Instance to Resources in a VPC \(p. 2493\)](#).
- **Launches an ML compute instance**—SageMaker launches an ML compute instance in a SageMaker VPC. SageMaker performs the configuration tasks that allow it to manage your notebook instance, and if you specified your VPC, it enables traffic between your VPC and the notebook instance.
- **Installs Anaconda packages and libraries for common deep learning platforms**—SageMaker installs all of the Anaconda packages that are included in the installer. For more information, see [Anaconda package list](#). In addition, SageMaker installs the TensorFlow and Apache MXNet deep learning libraries.
- **Attaches an ML storage volume**—SageMaker attaches an ML storage volume to the ML compute instance. You can use the volume as a working area to clean up the training dataset or to temporarily store validation, test, or other data. Choose any size between 5 GB and 16384 GB, in 1 GB increments, for the volume. The default is 5 GB. ML storage volumes are encrypted, so SageMaker can't determine the amount of available free space on the volume. Because of this, you can increase the volume size when you update a notebook instance, but you can't decrease the volume size. If you want to decrease the size of the ML storage volume in use, create a new notebook instance with the desired size.

Only files and data saved within the `/home/ec2-user/SageMaker` folder persist between notebook instance sessions. Files and data that are saved outside this directory are overwritten when the notebook instance stops and restarts. Each notebook instance's `/tmp` directory provides a minimum of 10 GB of storage in an instant store. An instance store is temporary, block-level storage that isn't persistent. When the instance is stopped or restarted, SageMaker deletes the directory's contents. This temporary storage is part of the root volume of the notebook instance.

- **Copies example Jupyter notebooks**— These Python code examples illustrate model training and hosting exercises using various algorithms and training datasets.

### To create a SageMaker notebook instance:

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Choose **Notebook instances**, then choose **Create notebook instance**.
3. On the **Create notebook instance** page, provide the following information:
  - a. For **Notebook instance name**, type a name for your notebook instance.
  - b. For **Notebook instance type**, choose an instance size suitable for your use case. For a list of supported instance types and quotas, see [Amazon SageMaker Service Quotas](#).
  - c. For **Elastic Inference**, choose an inference accelerator type to associate with the notebook instance if you plan to conduct inferences from the notebook instance, or choose **none**. For information about elastic inference, see [Use Amazon SageMaker Elastic Inference \(EI\) \(p. 1876\)](#).
  - d. (Optional) **Additional configuration** lets advanced users create a shell script that can run when you create or start the instance. This script, called a lifecycle configuration script, can be used to set the environment for the notebook or to perform other functions. For information, see [Customize a Notebook Instance Using a Lifecycle Configuration Script \(p. 124\)](#).
  - e. (Optional) **Additional configuration** also lets you specify the size, in GB, of the ML storage volume that is attached to the notebook instance. You can choose a size between 5 GB and 16,384 GB, in 1 GB increments. You can use the volume to clean up the training dataset or to temporarily store validation or other data.

- f. For **IAM role**, choose either an existing IAM role in your account that has the necessary permissions to access SageMaker resources or choose **Create a new role**. If you choose **Create a new role**, SageMaker creates an IAM role named `AmazonSageMaker-ExecutionRole-YYYYMMDDTHHmSS`. The Amazon managed policy `AmazonSageMakerFullAccess` is attached to the role. The role provides permissions that allow the notebook instance to call SageMaker and Amazon S3.
- g. For **Root access**, to enable root access for all notebook instance users, choose **Enable**. To disable root access for users, choose **Disable**. If you enable root access, all notebook instance users have administrator privileges and can access and edit all files on it.
- h. (Optional) **Encryption key** lets you encrypt data on the ML storage volume attached to the notebook instance using an Amazon Key Management Service (Amazon KMS) key. If you plan to store sensitive information on the ML storage volume, consider encrypting the information.
- i. (Optional) **Network** lets you put your notebook instance inside a Virtual Private Cloud (VPC). A VPC provides additional security and restricts access to resources in the VPC from sources outside the VPC. For more information on VPCs, see [Amazon VPC User Guide](#).

**To add your notebook instance to a VPC:**

- i. Choose the **VPC** and a **SubnetId**.
- ii. For **Security Group**, choose your VPC's default security group.
- iii. If you need your notebook instance to have internet access, enable direct internet access. For **Direct internet access**, choose **Enable**. Internet access can make your notebook instance less secure. For more information, see [Connect a Notebook Instance to Resources in a VPC \(p. 2493\)](#).
- j. (Optional) To associate Git repositories with the notebook instance, choose a default repository and up to three additional repositories. For more information, see [Associate Git Repositories with SageMaker Notebook Instances \(p. 133\)](#).
- k. Choose **Create notebook instance**.

In a few minutes, Amazon SageMaker launches an ML compute instance—in this case, a notebook instance—and attaches an ML storage volume to it. The notebook instance has a preconfigured Jupyter notebook server and a set of Anaconda libraries. For more information, see the [CreateNotebookInstance API](#).

4. When the status of the notebook instance is `InService`, in the console, the notebook instance is ready to use. Choose **Open Jupyter** next to the notebook name to open the classic Jupyter dashboard.

You can choose **Open JupyterLab** to open the JupyterLab dashboard. The dashboard provides access to your notebook instance and sample SageMaker notebooks that contain complete code walkthroughs. These walkthroughs show how to use SageMaker to perform common machine learning tasks. For more information, see [Example Notebooks \(p. 131\)](#). For more information, see [Control root access to a SageMaker notebook instance \(p. 2412\)](#).

For more information about Jupyter notebooks, see [The Jupyter notebook](#).

## Access Notebook Instances

To access your Amazon SageMaker notebook instances, choose one of the following options:

- Use the console.

Choose **Notebook instances**. The console displays a list of notebook instances in your account. To open a notebook instance with a standard Jupyter interface, choose **Open Jupyter** for that instance. To open a notebook instance with a JupyterLab interface, choose **Open JupyterLab** for that instance.



The console uses your sign-in credentials to send a [CreatePresignedNotebookInstanceUrl](#) API request to SageMaker. SageMaker returns the URL for your notebook instance, and the console opens the URL in another browser tab and displays the Jupyter notebook dashboard.

**Note**

The URL that you get from a call to [CreatePresignedNotebookInstanceUrl](#) is valid only for 5 minutes. If you try to use the URL after the 5-minute limit expires, you are directed to the Amazon Web Services Management Console sign-in page.

- Use the API.

To get the URL for the notebook instance, call the [CreatePresignedNotebookInstanceUrl](#) API and use the URL that the API returns to open the notebook instance.

Use the Jupyter notebook dashboard to create and manage notebooks and to write code. For more information about Jupyter notebooks, see <http://jupyter.org/documentation.html>.

## Update a Notebook Instance

After you create a notebook instance, you can update it using the SageMaker console and [UpdateNotebookInstance](#) API operation.

You can update the tags of a notebook instance that is `InService`. To update any other attribute of a notebook instance, its status must be `Stopped`.

**To update a notebook instance in the SageMaker console:**

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Choose **Notebook instances**.
3. Choose the notebook instance that you want to update by selecting the notebook instance **Name** from the list.
4. If your notebook **Status** is not `Stopped`, select the **Stop** button to stop the notebook instance.

When you do this, the notebook instance status changes to `Stopping`. Wait until the status changes to `Stopped` to complete the following steps.

5. Select the **Edit** button to open the **Edit notebook instance** page.

6. Update your notebook instance and select the **Update notebook instance** button at the bottom of the page when you are done to return to the notebook instances page. Your notebook instance status changes to **Updating**.

When the notebook instance update is complete, the status changes to **Stopped**.

## Customize a Notebook Instance Using a Lifecycle Configuration Script

To install packages or sample notebooks on your notebook instance, configure networking and security for it, or otherwise use a shell script to customize it, use a lifecycle configuration. A *lifecycle configuration* provides shell scripts that run only when you create the notebook instance or whenever you start one. When you create a notebook instance, you can create a new lifecycle configuration and the scripts it uses or apply one that you already have.

You can also use a lifecycle configuration script to access Amazon services from your notebook. For example, you can create a script that lets you use your notebook to control other Amazon resources, such as an Amazon EMR instance.

We maintain a public repository of notebook lifecycle configuration scripts that address common use cases for customizing notebook instances at <https://github.com/aws-samples/amazon-sagemaker-notebook-instance-lifecycle-configuration-samples>.

### Note

Each script has a limit of 16384 characters.

The value of the `$PATH` environment variable that is available to both scripts is `/usr/local/sbin:/usr/local/bin:/usr/bin:/usr/sbin:/sbin:/bin`. The working directory, which is the value of the `$PWD` environment variable, is `/`.

View CloudWatch Logs for notebook instance lifecycle configurations in log group `/aws/sagemaker/NotebookInstances` in log stream `[notebook-instance-name]/[LifecycleConfigHook]`.

Scripts cannot run for longer than 5 minutes. If a script runs for longer than 5 minutes, it fails and the notebook instance is not created or started. To help decrease the run time of scripts, try the following:

- Cut down on necessary steps. For example, limit which conda environments in which to install large packages.
- Run tasks in parallel processes.
- Use the `nohup` command in your script.

### To create a lifecycle configuration

1. For **Lifecycle configuration - Optional**, choose **Create a new lifecycle configuration**.
2. For **Name**, type a name using alphanumeric characters and `-`, but no spaces. The name can have a maximum of 63 characters.
3. (Optional) To create a script that runs when you create the notebook and every time you start it, choose **Start notebook**.
4. In the **Start notebook** editor, type the script.
5. (Optional) To create a script that runs only once, when you create the notebook, choose **Create notebook**.
6. In the **Create notebook** editor, type the script configure networking.
7. Choose **Create configuration**.

You can see a list of notebook instance lifecycle configurations you previously created by choosing **Lifecycle configuration** in the SageMaker console. From there, you can view, edit, delete existing lifecycle configurations. You can create a new notebook instance lifecycle configuration by choosing **Create configuration**. These notebook instance lifecycle configurations are available when you create a new notebook instance.

## Lifecycle Configuration Best Practices

The following are best practices for using lifecycle configurations:

- Lifecycle configurations run as the `root` user. If your script makes any changes within the `/home/ec2-user/SageMaker` directory, (for example, installing a package with `pip`), use the command `sudo -u ec2-user` to run as the `ec2-user` user. This is the same user that Amazon SageMaker runs as.
- SageMaker notebook instances use conda environments to implement different kernels for Jupyter notebooks. If you want to install packages that are available to one or more notebook kernels, enclose the commands to install the packages with conda environment commands that activate the conda environment that contains the kernel where you want to install the packages.

For example, if you want to install a package only for the `python3` environment, use the following code:

```
#!/bin/bash
sudo -u ec2-user -i <<'EOF'

# This will affect only the Jupyter kernel called "conda_python3".
source activate python3

# Replace myPackage with the name of the package you want to install.
pip install myPackage
# You can also perform "conda install" here as well.

source deactivate

EOF
```

If you want to install a package in all conda environments in the notebook instance, use the following code:

```
#!/bin/bash
sudo -u ec2-user -i <<'EOF'

# Note that "base" is special environment name, include it there as well.
for env in base /home/ec2-user/anaconda3/envs/*; do
    source /home/ec2-user/anaconda3/bin/activate $(basename "$env")

# Installing packages in the Jupyter system environment can affect stability of your
SageMaker
# Notebook Instance. You can remove this check if you'd like to install Jupyter
extensions, etc.
if [ $env = 'JupyterSystemEnv' ]; then
    continue
fi

# Replace myPackage with the name of the package you want to install.
pip install --upgrade --quiet myPackage
# You can also perform "conda install" here as well.

source /home/ec2-user/anaconda3/bin/deactivate
done
```

EOF

- You must store all conda environments in the default environments folder (/home/user/anaconda3/envs).

**Important**

When you create or change a script, we recommend that you use a text editor that provides Unix-style line breaks, such as the text editor available in the console when you create a notebook. Copying text from a non-Linux operating system might introduce incompatible line breaks and result in an unexpected error.

## Install External Libraries and Kernels in Notebook Instances

Amazon SageMaker notebook instances come with multiple environments already installed. These environments contain Jupyter kernels and Python packages including: scikit, Pandas, NumPy, TensorFlow, and MXNet. These environments, along with all files in the sample-notebooks folder, are refreshed when you stop and start a notebook instance. You can also install your own environments that contain your choice of packages and kernels.

The different Jupyter kernels in Amazon SageMaker notebook instances are separate conda environments. For information about conda environments, see [Managing environments](#) in the *Conda* documentation.

Install custom environments and kernels on the notebook instance's Amazon EBS volume. This ensures that they persist when you stop and restart the notebook instance, and that any external libraries you install are not updated by SageMaker. To do that, use a lifecycle configuration that includes both a script that runs when you create the notebook instance (on-create) and a script that runs each time you restart the notebook instance (on-start). For more information about using notebook instance lifecycle configurations, see [Customize a Notebook Instance Using a Lifecycle Configuration Script \(p. 124\)](#). There is a GitHub repository that contains sample lifecycle configuration scripts at [SageMaker Notebook Instance Lifecycle Config Samples](#).

The examples at <https://github.com/aws-samples/amazon-sagemaker-notebook-instance-lifecycle-config-samples/blob/master/scripts/persistent-conda-ebs/on-create.sh> and <https://github.com/aws-samples/amazon-sagemaker-notebook-instance-lifecycle-config-samples/blob/master/scripts/persistent-conda-ebs/on-start.sh> show the best practice for installing environments and kernels on a notebook instance. The on-create script installs the ipykernel library so that you can use create custom environments as Jupyter kernels, and then uses pip install and conda install to install libraries. You can adapt the script to create custom environments and install libraries that you want. SageMaker does not update these libraries when you stop and restart the notebook instance, so you can ensure that your custom environment has specific versions of libraries that you want. The on-start script installs any custom environments that you create as Jupyter kernels, so that they appear in the dropdown list in the Jupyter New menu.

## Package installation tools

SageMaker notebooks support the following package installation tools:

- conda install
- pip install

You can install packages using the following methods:

- Lifecycle configuration scripts.

For example scripts, see [SageMaker Notebook Instance Lifecycle Config Samples](#). For more information on lifecycle configuration, see [Customize a Notebook Instance Using a Lifecycle Configuration Script](#).

- Notebooks – The following commands are supported.
  - `%conda install`
  - `%pip install`
- The Jupyter terminal – You can install packages using pip and conda directly.

From within a notebook you can use the system command syntax (lines starting with !) to install packages, for example, `!pip install` and `!conda install`. More recently, new commands have been added to IPython: `%pip` and `%conda`. These commands are the recommended way to install packages from a notebook as they correctly take into account the activate environment or interpreter being used. For more information, see [Add %pip and %conda magic functions](#).

## Conda

Conda is an open source package management system and environment management system, which can install packages and their dependencies. SageMaker supports using Conda with either of the two main channels, the default channel, and the conda-forge channel. For more information, see [Conda channels](#). The conda-forge channel is a community channel where contributors can upload packages.

### Note

Due to how Conda resolves the dependency graph, installing packages from conda-forge can take significantly longer (in the worst cases, upwards of 10 minutes).

The Deep Learning AMI comes with many conda environments and many packages preinstalled. Due to the number of packages preinstalled, finding a set of packages that are guaranteed to be compatible is difficult. You may see a warning "The environment is inconsistent, please check the package plan carefully". Despite this warning, SageMaker ensures that all the SageMaker provided environments are correct. SageMaker cannot guarantee that any user installed packages will function correctly.

Conda has two methods for activating environments: `conda activate/deactivate`, and `source activate/deactivate`. For more information, see [Should I use 'conda activate' or 'source activate' in Linux](#).

SageMaker supports moving Conda environments onto the Amazon EBS volume, which is persisted when the instance is stopped. The environments aren't persisted when the environments are installed to the root volume, which is the default behavior. For an example lifecycle script, see [persistent-conda-ebs](#).

### Supported conda operations (see note at the bottom of this topic)

- `conda install` of a package in a single environment
- `conda install` of a package in all environments
- `conda install` of a R package in the R environment
- Installing a package from the main conda repository
- Installing a package from conda-forge
- Changing the Conda install location to use EBS
- Supporting both `conda activate` and `source activate`

## Pip

Pip is the de facto tool for installing and managing Python packages. Pip searches for packages on the Python Package Index (PyPI) by default. Unlike Conda, pip doesn't have built in environment support, and is not as thorough as Conda when it comes to packages with native/system library dependencies. Pip can be used to install packages in Conda environments.

You can use alternative package repositories with pip instead of the PyPI. For an example lifecycle script, see [on-start.sh](#).

### Supported pip operations (see note at the bottom of this topic)

- Using pip to install a package without an active conda environment (install packages system wide)
- Using pip to install a package in a conda environment
- Using pip to install a package in all conda environments
- Changing the pip install location to use EBS
- Using an alternative repository to install packages with pip

### Unsupported

SageMaker aims to support as many package installation operations as possible. However, if the packages were installed by SageMaker or DLAMI, and you use the following operations on these packages, it might make your notebook instance unstable:

- Uninstalling
- Downgrading
- Upgrading

We do not provide support for installing packages via yum install or installing R packages from CRAN.

Due to potential issues with network conditions or configurations, or the availability of Conda or PyPi, we cannot guarantee that packages will install in a fixed or deterministic amount of time.

#### Note

We cannot guarantee that a package installation will be successful. Attempting to install a package in an environment with incompatible dependencies can result in a failure. In such a case you should contact the library maintainer to see if it is possible to update the package dependencies. Alternatively you can attempt to modify the environment in such a way as to allow the installation. This modification however will likely mean removing or updating existing packages, which means we can no longer guarantee stability of this environment.

## Notebook Instance Software Updates

Amazon SageMaker periodically tests and releases software that is installed on notebook instances. This includes:

- Kernel updates
- Security patches
- Amazon SDK updates
- [Amazon SageMaker Python SDK](#) updates
- Open source software updates

SageMaker does not automatically update software on a notebook instance when it is in service. To ensure that you have the most recent software updates, stop and restart your notebook instance, either in the SageMaker console or by calling [StopNotebookInstance](#).

You can also manually update software installed on your notebook instance while it is running by using update commands in a terminal or in a notebook.

**Note**

Updating kernels and some packages might depend on whether root access is enabled for the notebook instance. For more information, see [Control root access to a SageMaker notebook instance \(p. 2412\)](#).

Notebook instances do not notify you if you are running outdated software. You can check the [Personal Health Dashboard](#) or the security bulletin at [Security Bulletins](#) for updates.

## Control an Amazon EMR Spark Instance Using a Notebook

You can use a notebook instance created with a custom lifecycle configuration script to access Amazon services from your notebook. For example, you can create a script that lets you use your notebook with Sparkmagic to control other Amazon resources, such as an Amazon EMR instance. You can then use the Amazon EMR instance to process your data instead of running the data analysis on your notebook. This allows you to create a smaller notebook instance because you won't use the instance to process data. This is helpful when you have large datasets that would require a large notebook instance to process the data.

The process requires three procedures using the Amazon SageMaker console:

- Create the Amazon EMR Spark instance
- Create the Jupyter Notebook
- Test the notebook-to-Amazon EMR connection

### To create an Amazon EMR Spark instance that can be controlled from a notebook using Sparkmagic

1. Open the Amazon EMR console at <https://console.amazonaws.cn/elasticmapreduce/>.
2. In the navigation pane, choose **Create cluster**.
3. On the **Create Cluster - Quick Options** page, under **Software configuration**, choose **Spark: Spark 2.4.4 on Hadoop 2.8.5 YARN with Ganglia 3.7.2 and Zeppelin 0.8.2**.
4. Set additional parameters on the page and then choose **Create cluster**.
5. On the **Cluster** page, choose the cluster name that you created. Note the **Master Public DNS**, the **EMR master's security group**, and the VPC name and subnet ID where the EMR cluster was created. You will use these values when you create a notebook.

### To create a notebook that uses Sparkmagic to control an Amazon EMR Spark instance

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the navigation pane, under **Notebook instances**, choose **Create notebook**.
3. Enter the notebook instance name and choose the instance type.
4. Choose **Additional configuration**, then, under **Lifecycle configuration**, choose **Create a new lifecycle configuration**.
5. Add the following code to the lifecycle configuration script:

```
# OVERVIEW
# This script connects an Amazon EMR cluster to an Amazon SageMaker notebook instance
# that uses Sparkmagic.
#
# Note that this script will fail if the Amazon EMR cluster's master node IP address is
# not reachable.
```

```

# 1. Ensure that the EMR master node IP is resolvable from the notebook instance.
# One way to accomplish this is to have the notebook instance and the Amazon EMR
# cluster in the same subnet.
# 2. Ensure the EMR master node security group provides inbound access from the
#    notebook instance security group.
#      Type      - Protocol - Port - Source
#      Custom TCP - TCP      - 8998 - $NOTEBOOK_SECURITY_GROUP
# 3. Ensure the notebook instance has internet connectivity to fetch the SparkMagic
#    example config.
#
# https://aws.amazon.com/blogs/machine-learning/build-amazon-sagemaker-notebooks-
# backed-by-spark-in-amazon-emr/
#
# PARAMETERS
EMR_MASTER_IP=your.emr.master.ip

cd /home/ec2-user/.sparkmagic

echo "Fetching Sparkmagic example config from GitHub..."
wget https://raw.githubusercontent.com/jupyter-incubator/sparkmagic/master/sparkmagic/
example_config.json

echo "Replacing EMR master node IP in Sparkmagic config..."
sed -i -- "s/localhost/$EMR_MASTER_IP/g" example_config.json
mv example_config.json config.json

echo "Sending a sample request to Livy.."
curl "$EMR_MASTER_IP:8998/sessions"

```

6. In the **PARAMETERS** section of the script, replace `your.emr.master.ip` with the Master Public DNS name for the Amazon EMR instance.
7. Choose **Create configuration**.
8. On the **Create notebook** page, choose **Network - optional**.
9. Choose the VPC and subnet where the Amazon EMR instance is located.
10. Choose the security group used by the Amazon EMR master node.
11. Choose **Create notebook instance**.

While the notebook instance is being created, the status is **Pending**. After the instance has been created and the lifecycle configuration script has successfully run, the status is **InService**.

#### Note

If the notebook instance can't connect to the Amazon EMR instance, SageMaker can't create the notebook instance. The connection can fail if the Amazon EMR instance and notebook are not in the same VPC and subnet, if the Amazon EMR master security group is not used by the notebook, or if the Master Public DNS name in the script is incorrect.

#### To test the connection between the Amazon EMR instance and the notebook

1. When the status of the notebook is **InService**, choose **Open Jupyter** to open the notebook.
2. Choose **New**, then choose **Sparkmagic (PySpark)**.
3. In the code cell, enter `%%info` and then run the cell.

The output should be similar to the following

```
Current session configs: {'driverMemory': '1000M', 'executorCores': 2, 'kind':
'pyspark'}
No active sessions.
```

## Example Notebooks

Your notebook instance contains example notebooks provided by Amazon SageMaker. The example notebooks contain code that shows how to apply machine learning solutions by using SageMaker. Notebook instances use the `nbexamples` Jupyter extension, which enables you to view a read-only version of an example notebook or create a copy of it so that you can modify and run it. For more information about the `nbexamples` extension, see <https://github.com/danielballan/nbexamples>. For information about example notebooks for SageMaker Studio, see [Use Amazon SageMaker Studio Notebooks \(p. 76\)](#).

### Note

Example notebooks typically download datasets from the internet. If you disable SageMaker-provided internet access when you create your notebook instance, example notebooks might not work. For more information, see [Connect a Notebook Instance to Resources in a VPC \(p. 2493\)](#).

## Use or View Example Notebooks in Jupyter Classic

To view or use the example notebooks in the classic Jupyter view, choose the **SageMaker Examples** tab.

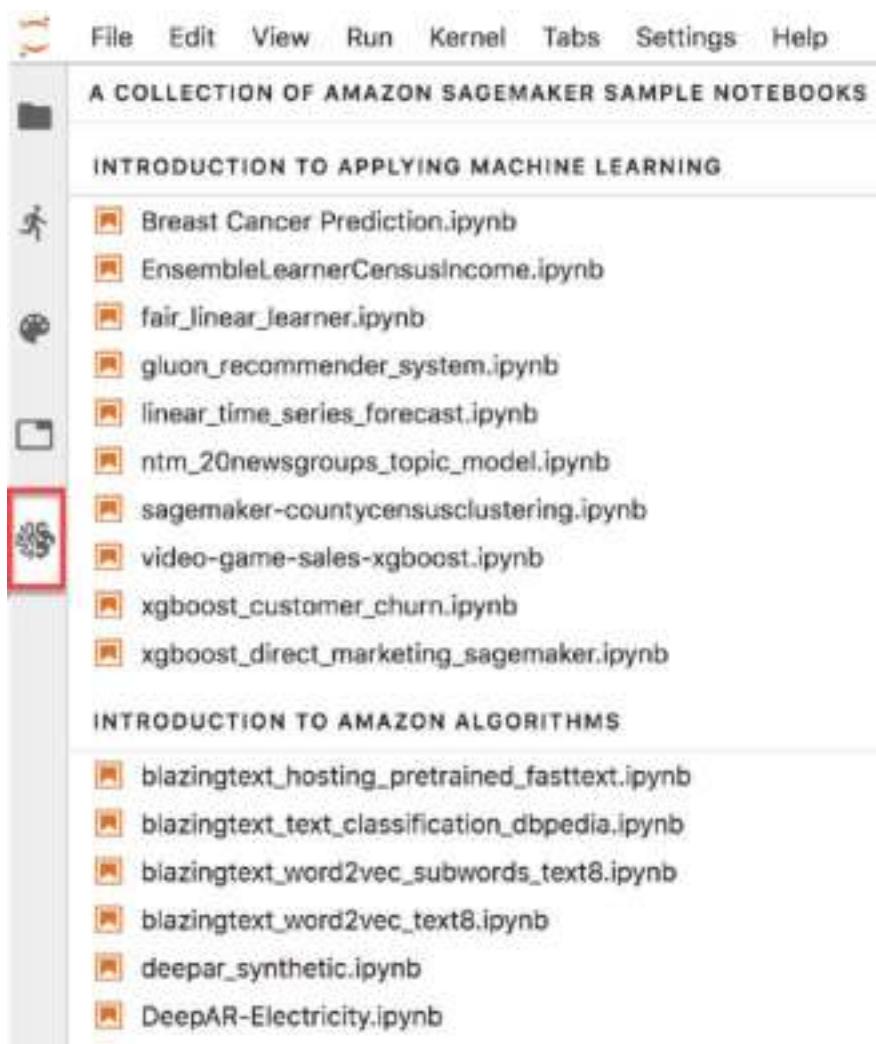


To view a read-only version of an example notebook in the Jupyter classic view, on the **SageMaker Examples** tab, choose **Preview** for that notebook. To create a copy of an example notebook in the home directory of your notebook instance, choose **Use**. In the dialog box, you can change the notebook's name before saving it.



## Use or View Example Notebooks in Jupyterlab

To view or use the example notebooks in the Jupyterlab view, choose the examples icon in the left navigation panel.



To view a read-only version of an example notebook, choose the name of the notebook. This opens the notebook as a tab in the main area. To create a copy of an example notebook in the home directory of your notebook instance, choose **Create a Copy** in the top banner. In the dialog box, type a name for the notebook and then choose **CREATE COPY**.

For more information about the example notebooks, see the [SageMaker examples GitHub repository](#).

## Set the Notebook Kernel

Amazon SageMaker provides several kernels for Jupyter that provide support for Python 2 and 3, Apache MXNet, TensorFlow, and PySpark. To set a kernel for a new notebook in the Jupyter notebook dashboard, choose **New**, and then choose the kernel from the list.



You can also create a custom kernel that you can use in your notebook instance. For information, see [Install External Libraries and Kernels in Notebook Instances \(p. 126\)](#).

## Associate Git Repositories with SageMaker Notebook Instances

Associate Git repositories with your notebook instance to save your notebooks in a source control environment that persists even if you stop or delete your notebook instance. You can associate one default repository and up to three additional repositories with a notebook instance. The repositories can be hosted in Amazon CodeCommit, GitHub, or on any other Git server. Associating Git repositories with your notebook instance can be useful for:

- Persistence - Notebooks in a notebook instance are stored on durable Amazon EBS volumes, but they do not persist beyond the life of your notebook instance. Storing notebooks in a Git repository enables you to store and use notebooks even if you stop or delete your notebook instance.
- Collaboration - Peers on a team often work on machine learning projects together. Storing your notebooks in Git repositories allows peers working in different notebook instances to share notebooks and collaborate on them in a source-control environment.
- Learning - Many Jupyter notebooks that demonstrate machine learning techniques are available in publicly hosted Git repositories, such as on GitHub. You can associate your notebook instance with a repository to easily load Jupyter notebooks contained in that repository.

There are two ways to associate a Git repository with a notebook instance:

- Add a Git repository as a resource in your Amazon SageMaker account. Then, to access the repository, you can specify an Amazon Secrets Manager secret that contains credentials. That way, you can access repositories that require authentication.
- Associate a public Git repository that is not a resource in your account. If you do this, you cannot specify credentials to access the repository.

### Topics

- [Add a Git Repository to Your Amazon SageMaker Account \(p. 134\)](#)
- [Create a Notebook Instance with an Associated Git Repository \(p. 136\)](#)
- [Associate a CodeCommit Repository in a Different Amazon Account with a Notebook Instance \(p. 137\)](#)
- [Use Git Repositories in a Notebook Instance \(p. 138\)](#)

# Add a Git Repository to Your Amazon SageMaker Account

To manage your GitHub repositories, easily associate them with your notebook instances, and associate credentials for repositories that require authentication, add the repositories as resources in your Amazon SageMaker account. You can view a list of repositories that are stored in your account and details about each repository in the SageMaker console and by using the API.

You can add Git repositories to your SageMaker account in the SageMaker console or by using the Amazon CLI.

## Note

You can use the SageMaker API

[CreateCodeRepository](#) to add Git repositories to your SageMaker account, but step-by-step instructions are not provided here.

## Add a Git Repository to Your SageMaker Account (Console)

### To add a Git repository as a resource in your SageMaker account

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Choose **Git repositories**, then choose **Add repository**.
3. To add an CodeCommit repository, choose **Amazon CodeCommit**. To add a GitHub or other Git-based repository, choose **GitHub/Other Git-based repo**.

### To add an existing CodeCommit repository

1. Choose **Use existing repository**.
2. For **Repository**, choose a repository from the list.
3. Enter a name to use for the repository in SageMaker. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
4. Choose **Add repository**.

### To create a new CodeCommit repository

1. Choose **Create new repository**.
2. Enter a name for the repository that you can use in both CodeCommit and SageMaker. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
3. Choose **Create repository**.

### To add a Git repository hosted somewhere other than CodeCommit

1. Choose **GitHub/Other Git-based repo**.
2. Enter a name of up to 63 characters. Valid characters include alpha-numeric characters, a hyphen (-), and 0-9.
3. Enter the URL for the repository. Do not provide a user name in the URL. Add the username and password in Amazon Secrets Manager as described in the next step.
4. For **Git credentials**, choose the credentials to use to authenticate to the repository. This is necessary only if the Git repository is private.

## Note

If you have two-factor authentication enabled for your Git repository, use a personal access token generated by your Git service provider instead of a password.

- a. To use an existing Amazon Secrets Manager secret, choose **Use existing secret**, and then choose a secret from the list. For information about creating and storing a secret, see [Creating a Basic Secret](#) in the *Amazon Secrets Manager User Guide*. The name of the secret you use must contain the string sagemaker.

**Note**

The secret must have a staging label of AWSCURRENT and must be in the following format:

```
{ "username": UserName, "password": Password}
```

For GitHub repositories, we recommend using a personal access token instead of your account password. For information, see <https://help.github.com/articles/creating-a-personal-access-token-for-the-command-line/>.

- b. To create a new Amazon Secrets Manager secret, choose **Create secret**, enter a name for the secret, and then enter the username and password to use to authenticate to the repository. The name for the secret must contain the string sagemaker.

**Note**

The IAM role you use to create the secret must have the secretsmanager:GetSecretValue permission in its IAM policy.

The secret must have a staging label of AWSCURRENT and must be in the following format:

```
{ "username": UserName, "password": Password}
```

For GitHub repositories, we recommend using a personal access token instead of your account password.

- c. To not use any credentials, choose **No secret**.

5. Choose **Create secret**.

## Add a Git Repository to Your Amazon SageMaker Account (CLI)

Use the `create-code-repository` Amazon CLI command. Specify a name for the repository as the value of the `code-repository-name` argument. The name must be 1 to 63 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen). Also specify the following:

- The default branch
- The URL of the Git repository

**Note**

Do not provide a user name in the URL. Add the username and password in Amazon Secrets Manager as described in the next step.

- The Amazon Resource Name (ARN) of an Amazon Secrets Manager secret that contains the credentials to use to authenticate the repository as the value of the `git-config` argument

For information about creating and storing a secret, see [Creating a Basic Secret](#) in the *Amazon Secrets Manager User Guide*. The following command creates a new repository named `MyRepository` in your Amazon SageMaker account that points to a Git repository hosted at <https://github.com/myprofile/my-repo>.

For Linux, OS X, or Unix:

```
aws sagemaker create-code-repository \
--code-repository-name "MyRepository" \
--git-config '{"Branch":"master", "RepositoryUrl" :
"https://github.com/myprofile/my-repo", "SecretArn" :
"arn:aws:secretsmanager:us-east-2:012345678901:secret:my-secret-ABC0DE"}'
```

For Windows:

```
aws sagemaker create-code-repository ^
--code-repository-name "MyRepository" ^
--git-config "{\"Branch\":\"master\", \"RepositoryUrl\": :"
\"https://github.com/myprofile/my-repo\", \"SecretArn\": :
\"arn:aws:secretsmanager:us-east-2:012345678901:secret:my-secret-ABC0DE\"}"
```

#### Note

The secret must have a staging label of `AWSCURRENT` and must be in the following format:

```
{"username": UserName, "password": Password}
```

For GitHub repositories, we recommend using a personal access token instead of your account password.

## Create a Notebook Instance with an Associated Git Repository

You can associate Git repositories with a notebook instance when you create the notebook instance by using the Amazon Web Services Management Console, or the Amazon CLI. If you want to use a CodeCommit repository that is in a different Amazon account than the notebook instance, set up cross-account access for the repository. For information, see [Associate a CodeCommit Repository in a Different Amazon Account with a Notebook Instance \(p. 137\)](#).

#### Topics

- [Create a Notebook Instance with an Associated Git Repository \(Console\) \(p. 136\)](#)
- [Create a Notebook Instance with an Associated Git Repository \(CLI\) \(p. 137\)](#)

## Create a Notebook Instance with an Associated Git Repository (Console)

### To create a notebook instance and associate Git repositories in the Amazon SageMaker console

1. Follow the instructions at [Step 1: Create an Amazon SageMaker Notebook Instance \(p. 53\)](#).
2. For **Git repositories**, choose Git repositories to associate with the notebook instance.
  - a. For **Default repository**, choose a repository that you want to use as your default repository. SageMaker clones this repository as a subdirectory in the Jupyter startup directory at `/home/ec2-user/SageMaker`. When you open your notebook instance, it opens in this repository. To choose a repository that is stored as a resource in your account, choose its name from the list. To add a new repository as a resource in your account, choose **Add a repository to SageMaker (opens the Add repository flow in a new window)** and then follow the instructions at [Create a Notebook Instance with an Associated Git Repository \(Console\) \(p. 136\)](#). To clone a public repository that is not stored in your account, choose **Clone a public Git repository to this notebook instance only**, and then specify the URL for that repository.
  - b. For **Additional repository 1**, choose a repository that you want to add as an additional directory. SageMaker clones this repository as a subdirectory in the Jupyter startup directory at `/home/ec2-user/SageMaker`. To choose a repository that is stored as a resource in your account, choose its name from the list. To add a new repository as a resource in your account, choose **Add a repository to SageMaker (opens the Add repository flow in a new window)** and then follow the instructions at [Create a Notebook Instance with an Associated Git Repository \(Console\) \(p. 136\)](#). To clone a repository that is not stored in your account, choose **Clone**

a public Git repository to this notebook instance only, and then specify the URL for that repository.

Repeat this step up to three times to add up to three additional repositories to your notebook instance.

## Create a Notebook Instance with an Associated Git Repository (CLI)

To create a notebook instance and associate Git repositories by using the Amazon CLI, use the `create-notebook-instance` command as follows:

- Specify the repository that you want to use as your default repository as the value of the `default-code-repository` argument. Amazon SageMaker clones this repository as a subdirectory in the Jupyter startup directory at `/home/ec2-user/SageMaker`. When you open your notebook instance, it opens in this repository. To use a repository that is stored as a resource in your SageMaker account, specify the name of the repository as the value of the `default-code-repository` argument. To use a repository that is not stored in your account, specify the URL of the repository as the value of the `default-code-repository` argument.
- Specify up to three additional repositories as the value of the `additional-code-repositories` argument. SageMaker clones this repository as a subdirectory in the Jupyter startup directory at `/home/ec2-user/SageMaker`, and the repository is excluded from the default repository by adding it to the `.git/info/exclude` directory of the default repository. To use repositories that are stored as resources in your SageMaker account, specify the names of the repositories as the value of the `additional-code-repositories` argument. To use repositories that are not stored in your account, specify the URLs of the repositories as the value of the `additional-code-repositories` argument.

For example, the following command creates a notebook instance that has a repository named `MyGitRepo`, that is stored as a resource in your SageMaker account, as a default repository, and an additional repository that is hosted on GitHub:

```
aws sagemaker create-notebook-instance \
    --notebook-instance-name "MyNotebookInstance" \
    --instance-type "ml.t2.medium" \
    --role-arn "arn:aws:iam::012345678901:role/service-role/
AmazonSageMaker-ExecutionRole-20181129T121390" \
    --default-code-repository "MyGitRepo" \
    --additional-code-repositories "https://github.com/myprofile/my-other-
repo"
```

### Note

If you use an Amazon CodeCommit repository that does not contain "SageMaker" in its name, add the `codecommit:GitPull` and `codecommit:GitPush` permissions to the role that you pass as the `role-arn` argument to the `create-notebook-instance` command. For information about how to add permissions to a role, see [Adding and Removing IAM Policies](#) in the *Amazon Identity and Access Management User Guide*.

## Associate a CodeCommit Repository in a Different Amazon Account with a Notebook Instance

To associate a CodeCommit repository in a different Amazon account with your notebook instance, set up cross-account access for the CodeCommit repository.

**To set up cross-account access for a CodeCommit repository and associate it with a notebook instance:**

1. In the Amazon account that contains the CodeCommit repository, create an IAM policy that allows access to the repository from users in the account that contains your notebook instance. For information, see [Step 1: Create a Policy for Repository Access in AccountA](#) in the *CodeCommit User Guide*.
2. In the Amazon account that contains the CodeCommit repository, create an IAM role, and attach the policy that you created in the previous step to that role. For information, see [Step 2: Create a Role for Repository Access in AccountA](#) in the *CodeCommit User Guide*.
3. Create a profile in the notebook instance that uses the role that you created in the previous step:
  - a. Open the notebook instance.
  - b. Open a terminal in the notebook instance.
  - c. Edit a new profile by typing the following in the terminal:

```
vi /home/ec2-user/.aws/config
```

- d. Edit the file with the following profile information:

```
[profile CrossAccountAccessProfile]
region = us-west-2
role_arn =
  arn:aws:iam::CodeCommitAccount:role/CrossAccountRepositoryContributorRole
credential_source=Ec2InstanceMetadata
output = json
```

Where `CodeCommitAccount` is the account that contains the CodeCommit repository, `CrossAccountAccessProfile` is the name of the new profile, and `CrossAccountRepositoryContributorRole` is the name of the role you created in the previous step.

4. On the notebook instance, configure git to use the profile you created in the previous step:
  - a. Open the notebook instance.
  - b. Open a terminal in the notebook instance.
  - c. Edit the Git configuration file typing the following in the terminal:

```
vi /home/ec2-user/.gitconfig
```

- d. Edit the file with the following profile information:

```
[credential]
  helper = !aws codecommit credential-helper --
profile CrossAccountAccessProfile $@
  UseHttpPath = true
```

Where `CrossAccountAccessProfile` is the name of the profile that you created in the previous step.

## Use Git Repositories in a Notebook Instance

When you open a notebook instance that has Git repositories associated with it, it opens in the default repository, which is installed in your notebook instance directly under `/home/ec2-user/SageMaker`.

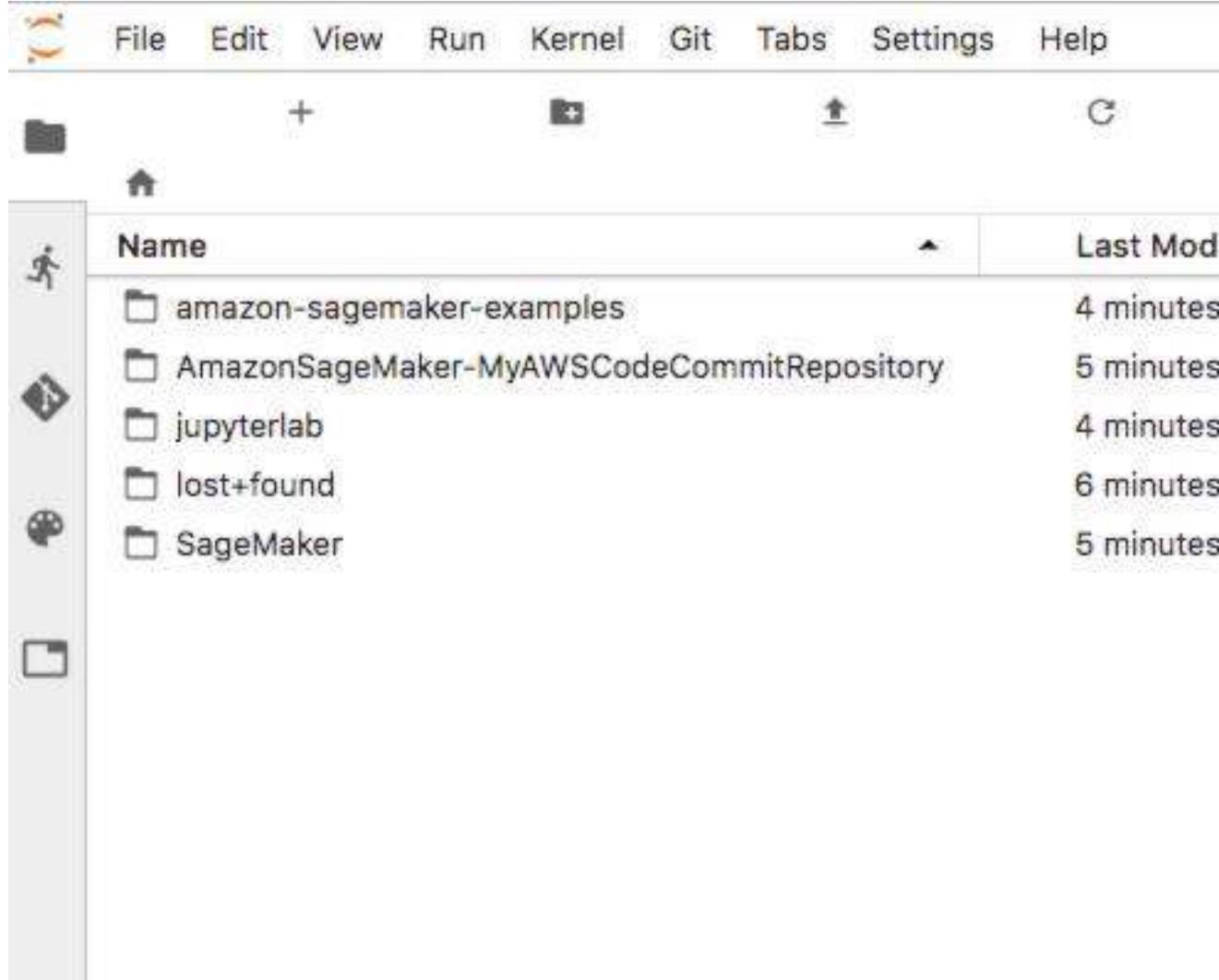
You can open and create notebooks, and you can manually run Git commands in a notebook cell. For example:

```
!git pull origin master
```

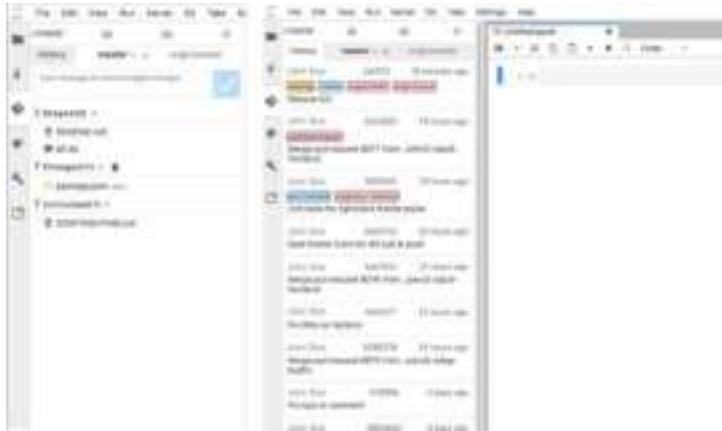
To open any of the additional repositories, navigate up one folder. The additional repositories are also installed as directories under /home/ec2-user/SageMaker.

If you open the notebook instance with a JupyterLab interface, the jupyter-git extension is installed and available to use. For information about the jupyter-git extension for JupyterLab, see <https://github.com/jupyterlab/jupyterlab-git>.

When you open a notebook instance in JupyterLab, you see the git repositories associated with it on the left menu:



You can use the jupyter-git extension to manage git visually, instead of using the command line:



## Notebook Instance Metadata

When you create a notebook instance, Amazon SageMaker creates a JSON file on the instance at the location `/opt/ml/metadata/resource-metadata.json` that contains the `ResourceName` and `ResourceArn` of the notebook instance. You can access this metadata from anywhere within the notebook instance, including in lifecycle configurations. For information about notebook instance lifecycle configurations, see [Customize a Notebook Instance Using a Lifecycle Configuration Script \(p. 124\)](#).

The `resource-metadata.json` file has the following structure:

```
{  
    "ResourceArn": "NotebookInstanceArn",  
    "ResourceName": "NotebookInstanceName"  
}
```

You can use this metadata from within the notebook instance to get other information about the notebook instance. For example, the following commands get the tags associated with the notebook instance:

```
NOTEBOOK_ARN=$(jq '.ResourceArn'  
    /opt/ml/metadata/resource-metadata.json --raw-output)  
aws sagemaker list-tags --resource-arn $NOTEBOOK_ARN
```

The output looks like the following:

```
{  
    "Tags": [  
        {  
            "Key": "test",  
            "Value": "true"  
        }  
    ]  
}
```

## Monitor Jupyter Logs in Amazon CloudWatch Logs

Jupyter logs include important information such as events, metrics, and health information that provide actionable insights when running Amazon SageMaker notebooks. By importing Jupyter logs into

CloudWatch Logs, customers can use CloudWatch Logs to detect anomalous behaviors, set alarms, and discover insights to keep the SageMaker notebooks running more smoothly. You can access the logs even when the Amazon EC2 instance that hosts the notebook is unresponsive, and use the logs to troubleshoot the unresponsive notebook. Sensitive information such as Amazon account IDs, secret keys, and authentication tokens in presigned URLs are removed so that customers can share logs without leaking private information.

**To view Jupyter logs for a notebook instance:**

1. Sign in to the Amazon Web Services Management Console and open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Choose **Notebook instances**.
3. In the list of notebook instances, choose the notebook instance for which you want to view Jupyter logs by selecting the Notebook instance **Name**.

This will bring you to the details page for that notebook instance.

4. Under **Monitor** on the notebook instance details page, choose **View logs**.
5. In the CloudWatch console, choose the log stream for your notebook instance. Its name is in the form *NotebookInstanceName/jupyter.log*.

For more information about monitoring CloudWatch logs for SageMaker, see [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 2534\)](#).

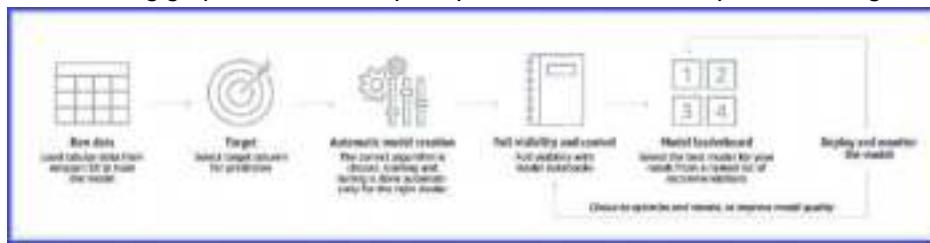
# Automate model development with Amazon SageMaker Autopilot

Amazon SageMaker Autopilot is a feature-set that automates key tasks of an automatic machine learning (AutoML) process. It explores your data, selects the algorithms relevant to your problem type, and prepares the data to facilitate model training and tuning. Autopilot applies a cross-validation resampling procedure automatically to all candidate algorithms when appropriate to test their ability to predict data they have not been trained on. It simplifies your machine learning experience by automating these key tasks that constitute an AutoML process. It ranks all of the optimized models tested by their performance. It finds the best performing model that you can deploy at a fraction of the time normally required.

Autopilot also helps explain how models make predictions using a feature attribution approach developed for Amazon SageMaker Clarify. Autopilot automatically generates a report that indicate the importance of each feature for the predictions made by the best candidate. This explainability functionality can make machine learning models more understandable to Amazon customers. The model governance report generated can be used to inform risk and compliance teams and external regulators.

You get full visibility into how the data was wrangled and how the models were selected, trained and tuned for each of the candidates tested. This is provided by notebooks that Autopilot generates for each trial that contain the code used to explore the data and find the best candidate. The notebooks also provide educational tools that enable you to learn about and conduct your own ML experiments. You can learn about the impact of various inputs and trade-offs made in experiments by examining the various data exploration and candidate definition notebooks exposed by Autopilot. You can also conduct further experiments on the higher performing candidates by making your own modifications to the notebooks and rerunning them.

The following graphic outlines the principal tasks of an AutoML process managed by Autopilot.



You can use Autopilot in different ways: on autopilot (hence the name) or with various degrees of human guidance, without code through Amazon SageMaker Studio, or with code using one of the Amazon SDKs. Autopilot currently supports regression and binary and multiclass classification. It also only supports tabular data formatted in files with comma-separated values.

With Amazon SageMaker, you pay only for what you use. Building, training, and deploying ML models is billed by the second, with no minimum fees and no upfront commitments. For more information about the cost of using SageMaker, see [Amazon SageMaker Pricing](#).

## Topics

- [Get started with Amazon SageMaker Autopilot \(p. 143\)](#)
- [Create an Amazon SageMaker Autopilot experiment \(p. 145\)](#)
- [Amazon SageMaker Autopilot problem types \(p. 148\)](#)
- [Model support and validation \(p. 149\)](#)

- [Amazon SageMaker Autopilot model deployment \(p. 150\)](#)
- [Amazon SageMaker Autopilot explainability \(p. 152\)](#)
- [Models generated by Amazon SageMaker Autopilot \(p. 153\)](#)
- [Amazon SageMaker Autopilot notebooks generated to manage AutoML tasks \(p. 156\)](#)
- [Configure inference output in Autopilot-generated containers \(p. 157\)](#)
- [Amazon SageMaker Autopilot quotas \(p. 159\)](#)
- [API reference guide for Amazon SageMaker Autopilot \(p. 161\)](#)

## Get started with Amazon SageMaker Autopilot

Amazon SageMaker Autopilot provides samples, videos, and tutorials to get started with Amazon SageMaker Autopilot

### Topics

- [Samples: Explore modeling with Amazon SageMaker Autopilot \(p. 143\)](#)
- [Videos: Use Autopilot to automate and explore the machine learning process \(p. 144\)](#)
- [Tutorials: Get started with Amazon SageMaker Autopilot \(p. 145\)](#)

## Samples: Explore modeling with Amazon SageMaker Autopilot

Amazon SageMaker Autopilot provides the following sample notebooks.

- [Direct marketing with Amazon SageMaker Autopilot](#): This notebook demonstrates how uses the [Bank Marketing Data Set](#) to predict whether a customer will enroll for a term deposit at a bank. You can use Autopilot on this dataset to get the most accurate ML pipeline by exploring options contained in various candidate pipelines. Autopilot generates each candidate in a two step procedure. The first step performs automated feature engineering on the dataset. The second step trains and tunes an algorithm to produce a model. The notebook contains instructions on how to train the model as well as how to deploy the model to perform batch inference using the best candidate.
- [Customer Churn Prediction with Amazon SageMaker Autopilot](#): This notebook describes using machine learning for the automated identification of unhappy customers, also known as customer churn prediction. The sample shows how to analyze a publicly available dataset and perform feature engineering on it. Next it shows how to tune a model by selecting the best performing pipeline along with the optimal hyperparameters for the training algorithm. Finally it shows how to deploy the model to a hosted endpoint and evaluate its predictions against ground truth. ML models rarely give perfect predictions though, so this notebook is also about how to incorporate the relative costs of prediction mistakes when determining the financial outcome of using ML.
- [Top Candidates Customer Churn Prediction with Amazon SageMaker Autopilot and Batch Transform \(Python SDK\)](#): This notebook also describes using machine learning for the automated identification of unhappy customers, also known as customer churn prediction. This notebook demonstrate how to configure the model to obtain the inference probability, select the top N models, and make Batch Transform on a hold-out test set for evaluation.

### Note

This notebook works with SageMaker Python SDK >= 1.65.1 released on 6/19/2020.

- [Bringing your own data processing code to SageMaker Autopilot](#): This notebook demonstrates how to incorporate and deploy custom data processing code when using Amazon SageMaker Autopilot. It adds a custom feature selection step to remove irrelevant variables to a Autopilot job. It then shows how to deploy both the custom processing code and models generated by Autopilot on a real-time endpoint and, alternatively, for batch processing.

## Videos: Use Autopilot to automate and explore the machine learning process

Here is a video series that provides a tour of Amazon SageMaker Autopilot capabilities using Studio. They show how to start an AutoML job, analyze and preprocess data, how to do feature engineering and hyperparameter optimization on candidate models, and how to visualize and compare the resulting model metrics.

### Topics

- [Start an AutoML job with Amazon SageMaker Autopilot \(p. 144\)](#)
- [Review data exploration and feature engineering automated in Autopilot. \(p. 144\)](#)
- [Tune models to optimize performance \(p. 144\)](#)
- [Choose and deploy the best model \(p. 144\)](#)
- [Amazon SageMaker Autopilot walkthrough \(p. 144\)](#)

### Start an AutoML job with Amazon SageMaker Autopilot

This video shows you how to start an AutoML job with Autopilot. (Length: 8:41)

[Amazon SageMaker Studio - AutoML with Amazon SageMaker Autopilot \(part 1\)](#)

### Review data exploration and feature engineering automated in Autopilot.

This video shows you how to examine the data exploration and candidate definition notebooks generated by Amazon SageMaker Autopilot. (Length: 10:04)

[Amazon SageMaker Studio - AutoML with Amazon SageMaker Autopilot \(part 2\)](#)

### Tune models to optimize performance

This video shows you how to optimize model performance during training using hyperparameter tuning. (Length: 4:59)

[SageMaker Studio - AutoML with Amazon SageMaker Autopilot \(part 3\)](#)

### Choose and deploy the best model

This video shows you how to use job metrics to choose the best model and then how to deploy it. (Length: 5:20)

[SageMaker Studio - AutoML with Amazon SageMaker Autopilot \(part 4\)](#)

### Amazon SageMaker Autopilot walkthrough

This video walks you through an end to end demo where we first build a binary classification model automatically with Amazon SageMaker Autopilot. We see how candidate models have been built and optimized using auto-generated notebooks. We also look at the top candidates with SageMaker Experiments. Finally, we deploy the top candidate (based on XGBoost), and configure data capture with SageMaker Model Monitor.

[End to end demo with AutoML on SageMaker](#)

## Tutorials: Get started with Amazon SageMaker Autopilot

Autopilot get started tutorials demonstrate how to create a machine learning model automatically without having to write code. They show you how Autopilot simplifies the machine learning experience by helping you explore your data and try different algorithms. Autopilot builds the best machine learning model for the problem type using AutoML capabilities while allowing full control and visibility.

- **Create a machine learning model automatically with Autopilot:** You assume the role of a developer working at a bank in this tutorial. You have been asked to develop a machine learning model to predict whether or not a customer will enroll for a certificate of deposit (CD). This is a binary classification problem. The model is trained on the marketing dataset that contains information on customer demographics, responses to marketing events, and external factors.

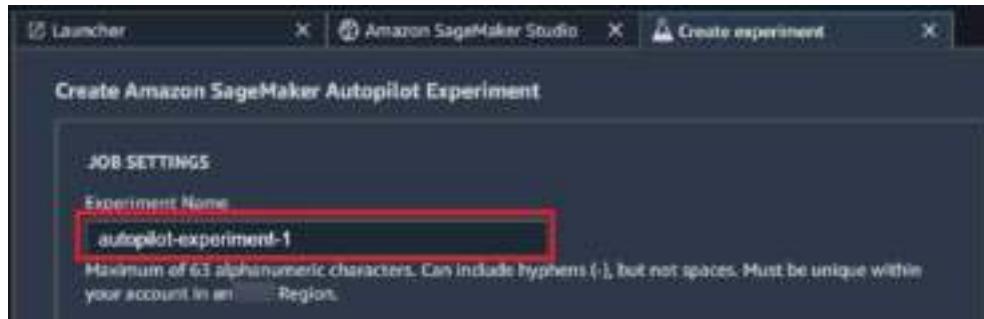
## Create an Amazon SageMaker Autopilot experiment

To create an Amazon SageMaker Autopilot experiment, you need to name it, provide locations for the input and output data, specify the target data to predict, and optionally the type of machine learning problem to solve. When you create an Amazon SageMaker Autopilot job as a pilot experiment, SageMaker analyzes your data and creates a notebook with candidate model definitions. If you choose to run the complete experiment option, SageMaker also trains and tunes these models on your behalf. You can view statistics while the experiment is running. Afterwards, you can compare trials and delve into the details.

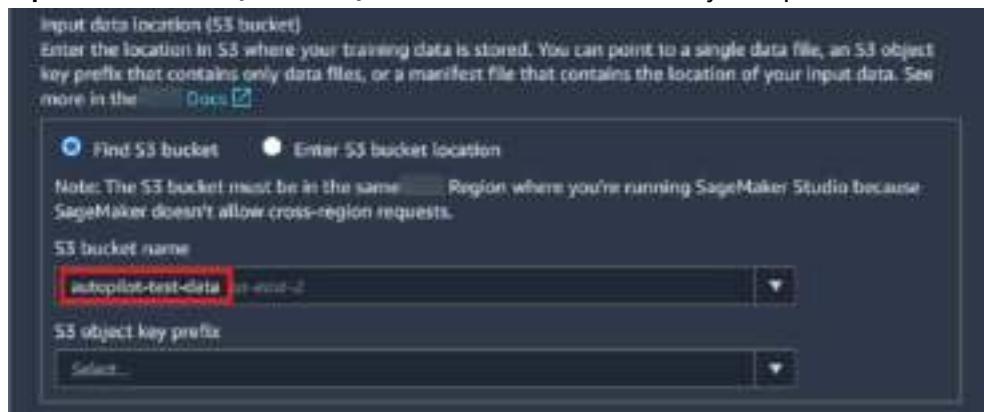
1. Open Amazon SageMaker Studio and sign in.
2. Choose the **Create Autopilot experiment** option from the **Build model automatically** box.



3. Enter information about the experiment in the **Job Settings** form:
  - **Experiment Name** – Must be unique to your account in the current Amazon Region and contain a maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces.



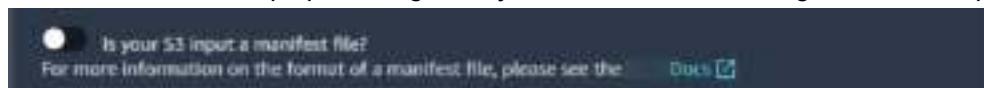
- **Input data location (S3 bucket)** – The S3 bucket that contains your input data.



#### Note

Must be an s3:// formatted URL where Amazon SageMaker has write permissions. The S3 bucket must be in the current Amazon Region and must be in CSV format and contain at least 500 rows.

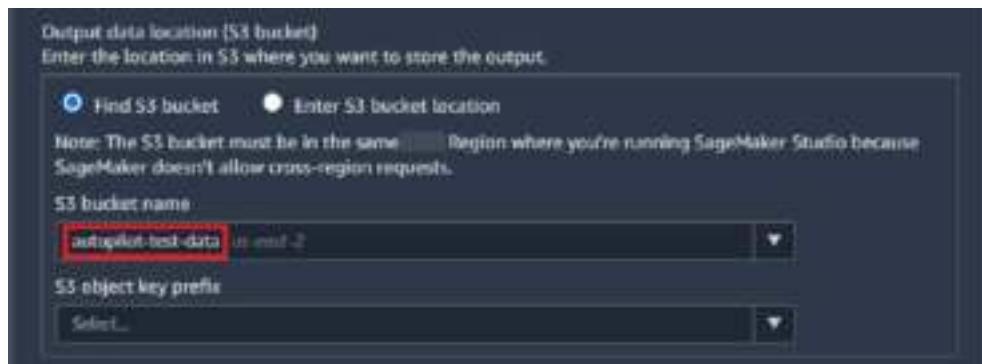
- **S3 bucket name** – The bucket name must be unique across all existing bucket names in S3.
- **S3 object key prefix** – The filename of the object in the bucket, including the path to the object within the bucket.
- **S3 bucket location** – The concatenation of the S3 bucket name and S3 object key prefix.
- **Is your S3 input a manifest file?** – A manifest file includes metadata with your input data. The metadata specifies the location of your data in Amazon S3 storage, how the data are formatted, and which attributes from the dataset to use when training your model. You can use a manifest file as an alternative to preprocessing when you have labeled data being streamed in Pipe mode.



- **Target attribute name** – The name of the data column you want the model to target for predictions.



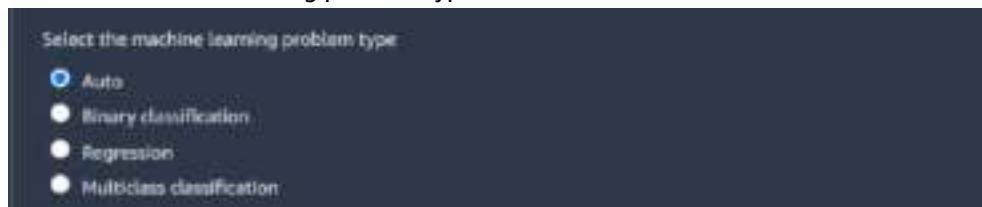
- **Output data location (S3 bucket)** – The S3 bucket where you want to store the output data.



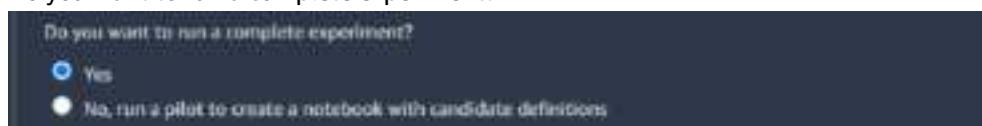
**Note**

Must be an s3:// formatted URL where Amazon SageMaker has write permissions. The S3 bucket must be in the current Amazon Region.

- **S3 bucket name** – The bucket name must be unique across all existing bucket names in S3.
- **S3 object key prefix** – The filename of the object in the bucket, including the path to the object within the bucket.
- **S3 bucket location** – The concatenation of the S3 bucket name and S3 object key prefix.
- Select the machine learning problem type:



- Auto – SageMaker infers the problem type from the values of the attribute you want to predict. In some cases, SageMaker is unable to infer accurately, in which case you must provide the value for the job to succeed.
- Binary classification – Binary classification is a type of supervised learning that assigns an individual to one of two predefined and mutually exclusive classes based on their attributes. For example, a medical diagnosis for whether an individual has a disease or not based on the results of diagnostic tests.
- Regression – Regression estimates the values of a dependent target variable based on one or more other variables or attributes that are correlated with it. For example, house prices based on features such as square footage and number of bathrooms.
- Multiclass classification – Multiclass classification is a type of supervised learning that assigns an individual to one of several classes based on their attributes. For example, the prediction of the topic most relevant to a text document, such as politics, finance, or philosophy.
- Do you want to run a complete experiment?

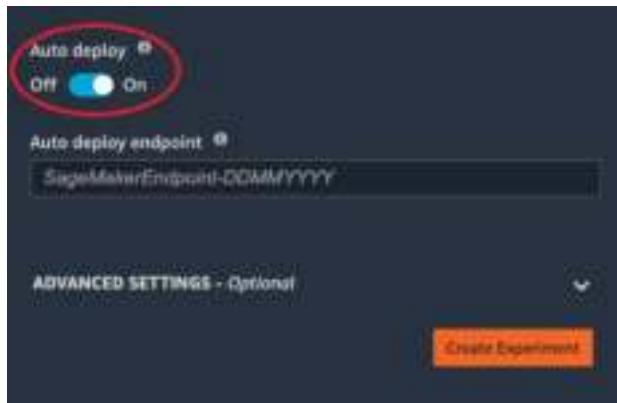


If you choose **Yes**, SageMaker generates a model as well as statistics that you can view in real time while the experiment is running. After the experiment is complete, you can view the trials, sort by objective metric, and right-click to deploy the model for use in other environments.

If you choose **No**, instead of executing the entire workflow, SageMaker stops execution after generating a notebook with candidate definitions. A candidate is a combination of data preprocessors, algorithms, and algorithm parameter settings. You can use the notebook as a

starting point to guide your own process of model training/tuning. The notebook has highlighted sections that explain what kinds of changes are typical, such as changing instance type, cluster size, and so on.

4. To automatically deploy the best model from an Autopilot experiment to an endpoint, accept the default **Auto deploy** value **On** when creating the experiment.



Choose **Create Experiment**.

**Note**

Automatic deployment will fail if the default resource quota or your customer quota for endpoint instances in a region are too limited. Currently the requirement is that you need have at least two ml.m5.2xlarge instances. The eu-north-1 region (Stockholm) does not meet this requirement, for example. The supported instance types for this region are listed at [SageMaker Instance Types in EU \(Stockholm\) eu-north-1](#). If you encounter this issue, you can request a service limit increase for SageMaker endpoints instances by following the procedure at [Supported Regions and Quotas \(p. 32\)](#). In the **Case details** panel, select **SageMaker Endpoints** for the **Limit type**. For **Request1**, select:

- **Region: EU (Stockholm)**
- **Resource Type: SageMaker Hosting**
- **Limit: ml.m5.2xlarge (at least)**
- **New limit value: 2**

**Note**

To avoid incurring unnecessary charges, delete the endpoints and resources that were created when deploying the model after they are no longer needed. Information on pricing of instances by region is available at [Amazon SageMaker Pricing](#).

## Amazon SageMaker Autopilot problem types

When setting a problem type, such as binary classification or regression, with the AutoML API, you have the option of specifying it or of letting Amazon SageMaker Autopilot detect it on your behalf. You set the type of problem with the `CreateAutoPilot.ProblemType` parameter. This limits the kind of preprocessing and algorithms that Autopilot tries. When the job is finished, if you had set the `CreateAutoPilot.ProblemType`, then the `ResolvedAttribute.ProblemType` will match the `ProblemType` you set. If you leave it blank (or `null`), the `ProblemType` will be whatever Autopilot decides on your behalf.

**Note**

In some cases, Autopilot is unable to infer the `ProblemType` with high enough confidence, in which case you must provide the value for the job to succeed.

Your problem type options are as follows:

**Topics**

- [Regression \(p. 149\)](#)
- [Binary classification \(p. 149\)](#)
- [Multiclass classification \(p. 149\)](#)

## Regression

Regression estimates the values of a dependent target variable based on one or more other variables or attributes that are correlated with it. An example is the prediction of house prices using features like the number of bathrooms and bedrooms, square footage of the house and garden. Regression analysis can create a model that takes one or more of these features as an input and predicts the price of a house.

## Binary classification

Binary classification is a type of supervised learning that assigns an individual to one of two predefined and mutually exclusive classes based on their attributes. It is supervised because the models are trained using examples where the attributes are provided with correctly labelled objects. A medical diagnosis for whether an individual has a disease or not based on the results of diagnostic tests is an example of binary classification.

## Multiclass classification

Multiclass classification is a type of supervised learning that assigns an individual to one of several classes based on their attributes. It is supervised because the models are trained using examples where the attributes are provided with correctly labelled objects. An example is the prediction of the topic most relevant to a text document. A document may be classified as being about, say, religion or politics or finance, or about one of several other predefined topic classes.

## Model support and validation

Amazon SageMaker Autopilot supports three types of machine learning algorithms to address machine learning problems and uses cross-validation automatically when needed.

**Topics**

- [Autopilot algorithm support \(p. 149\)](#)
- [Autopilot cross-validation \(p. 150\)](#)

## Autopilot algorithm support

The three types of machine learning algorithms supported by Autopilot are:

- [Linear Learner Algorithm \(p. 1442\)](#): a supervised learning algorithms used for solving either classification or regression problems.
- [XGBoost Algorithm \(p. 1524\)](#): a supervised learning algorithm that attempts to accurately predict a target variable by combining an ensemble of estimates from a set of simpler and weaker models.
- Deep Learning Algorithm: multilayer perceptron (MLP), a feedforward artificial neural network that can handle data that is not linear separable.

**Note**

You do not need to specify an algorithm to use for your machine learning problem. Autopilot automatically selects the appropriate algorithm to train.

## Autopilot cross-validation

Autopilot uses the k-fold cross-validation method automatically when needed. You can use this method to assess how well a model trained on a dataset can predict the values of an unseen validation dataset drawn from the same population. This method is especially important, for example, when training on datasets that have a limited number of training instances. It can protect against problems like overfitting and selection bias that can prevent a model from being more generally applicable to the population sampled.

For example, the Boston Housing dataset contains only 861 samples. If you try to build a model to predict house sale prices using this dataset without cross-validation, you risk training on a data set that is not representative of the Boston housing stock. Typically, you would split the data only once into training and validation subsets. If the training fold happened to contain data mainly from suburbs that were not representative of the rest of the city, you would likely overfit on this biased selection. Cross-validation reduces the risk of these errors by making full and randomized use of the available data for training and validation.

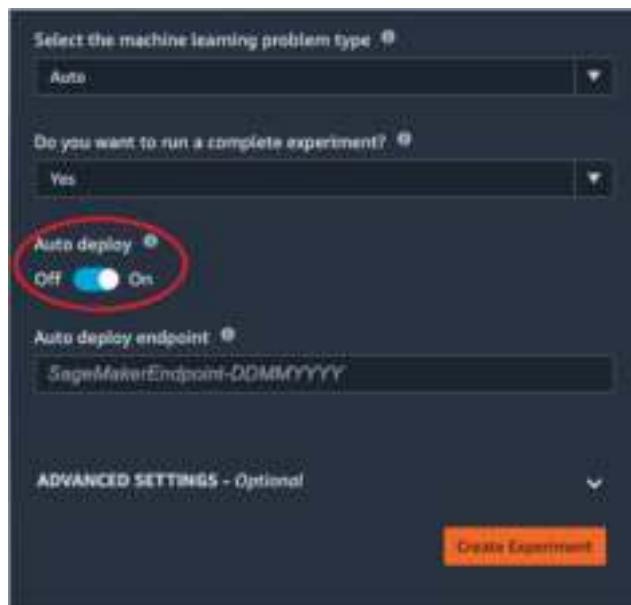
K-fold cross-validation randomly splits a training dataset into  $k$  equally sized subsamples or folds. Then models are trained on  $k-1$  folds and tested against the remaining fold, which is retained as a validation dataset. The process is repeated  $k$  times, using each fold once as the validation dataset. Autopilot applies the cross-validation method to datasets with 50,000 or fewer training instances. It uses a  $k$  value of 5 on the candidate algorithms used to model the dataset. Multiple models are trained on different splits and the models are stored separately. When the training procedure is finished, the validation metrics for each of the models are averaged to produce a single estimation metric. When cross-validation is applied by Autopilot to smaller datasets, the training time increases 20% on average. If your dataset is complicated, the training time may increase more significantly. For predictions, Autopilot uses the ensemble of cross-validation models from the trial with the best validation metric.

You can see the training and validation metrics from each fold in your `/aws/sagemaker/TrainingJobs` CloudWatch Logs. For more information about CloudWatch Logs, see [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 2534\)](#). The validation metric for the models trained by Autopilot is presented as the objective metric in model leader board. Autopilot uses the default validation metric for each problem type it handles unless you specify otherwise. For more information on the metrics Autopilot uses, see [AutoMLJobObjective](#). You can deploy Autopilot models built using cross-validation just as you would any other Autopilot or SageMaker model.

## Amazon SageMaker Autopilot model deployment

To deploy the model that produced the best validation metric in an Autopilot experiment, you have several options. When using Autopilot in SageMaker Studio, you can deploy the model automatically or manually. When working in another development, you can call Autopilot APIs directly to deploy a model.

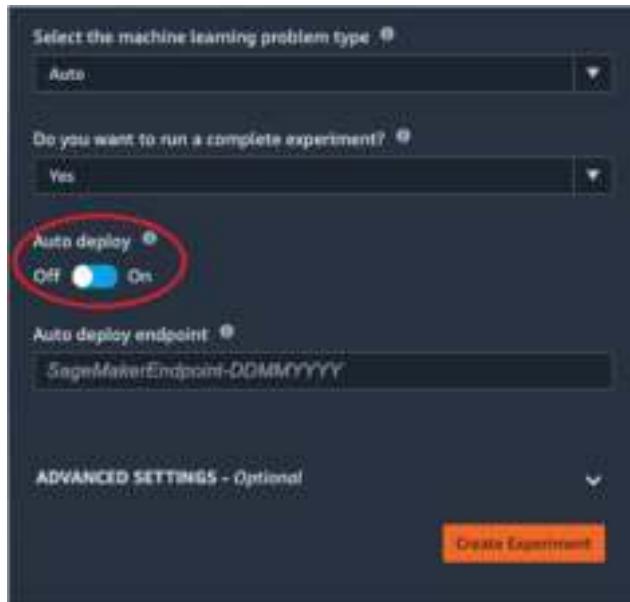
- **Automatically:** To automatically deploy the best model from an Autopilot experiment to an endpoint, accept the default **Auto deploy** value **On** when creating the experiment in SageMaker Studio.



**Note**

Automatic deployment will fail if the default resource quota or your customer quota for endpoint instances in a region are too limited. Currently the requirement is that you need have at least two ml.m5.2xlarge instances. The eu-north-1 region (Stockholm) does not meet this requirement, for example. The supported instance types for this region are listed at [SageMaker Instance Types in EU \(Stockholm\) eu-north-1](#). If you encounter this issue, you can request a service limit increase for SageMaker endpoints instances by following the procedure at [Supported Regions and Quotas \(p. 32\)](#). In the **Case details** panel, select **SageMaker Endpoints** for the **Limit type**. For **Request1**, select:

- **Region: EU (Stockholm)**
- **Resource Type: SageMaker Hosting**
- **Limit: ml.m5.2xlarge (at least)**
- **New limit value: 2**
- **Manually:** To manually deploy the best model from an Autopilot experiment to an endpoint, set the **Auto deploy** value to **Off** when creating the experiment in SageMaker Studio.



- **API calls:** Make the following series of API calls:
  1. [CreateAutoMLJob](#)
  2. [DescribeAutoMLJob](#)
  3. [ListCandidatesForAutoMLJob](#)
  4. [CreateModel](#)
  5. [CreateEndpointConfig](#)
  6. [CreateEndpoint](#)

The automatic deployment for the results of an experiment in SageMaker Studio calls the six APIs listed in this last option by default. For information on how to create an experiment, see [Create an Amazon SageMaker Autopilot experiment \(p. 145\)](#).

**Note**

To avoid incurring unnecessary charges, delete the endpoints and resources that were created when deploying the model after they are no longer needed. Information on pricing of instances by region is available at [Amazon SageMaker Pricing](#).

## Amazon SageMaker Autopilot explainability

Amazon SageMaker Autopilot uses tools provided by Amazon SageMaker Clarify to help explain how machine learning (ML) models make predictions. These tools can help ML modelers and developers and other internal stakeholders understand model characteristics as a whole prior to deployment and debug predictions provided by a model after it's deployed. Transparency about how ML models arrive at their predictions is also critical to consumers and regulators, who need to trust the model predictions if they are going to accept the decisions based on them. The Autopilot explanatory functionality uses a model-agnostic feature attribution approach, which you can use to understand why a model made a prediction after training and to provide per-instance explanation during inference. The implementation includes a scalable and efficient implementation of [SHAP](#), based on the concept of a Shapley value from the field of cooperative game theory that assigns each feature an importance value for a particular prediction.

You can use explanations for auditing and meeting regulatory requirements, building trust in the model and supporting human decision-making, and debugging and improving model performance.

For additional information on Shapely values and baselines, see [Feature Attributions that Use Shapley Values \(p. 1853\)](#) and [SHAP Baselines for Explainability \(p. 1854\)](#).

For a guide to the Amazon SageMaker Clarify documentation, see [Guide to the SageMaker Clarify Documentation \(p. 8\)](#).

## Models generated by Amazon SageMaker Autopilot

This procedure describes how to view details about Amazon SageMaker Autopilot jobs that you have run. Details provided about the candidate models generated by Autopilot include:

- A plot of the aggregated SHAP values that indicate the importance of each feature to help explain your models predictions.
- The summary statistics for various training and validation metrics, including the objective metric.
- A list of the hyperparameters used to train and tune the model.

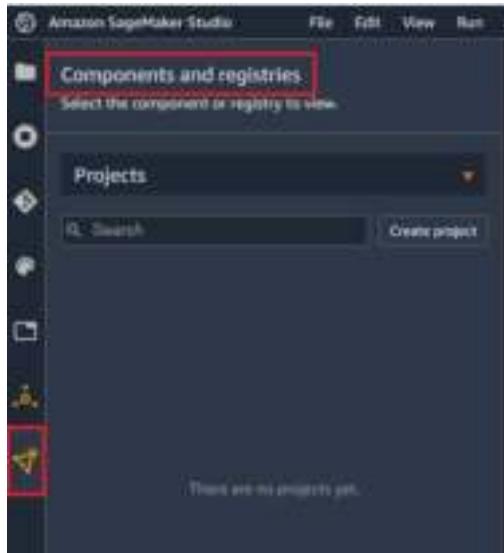
### Note

This topic assumes that you have already created and run an Autopilot experiment. For information on how to create an Autopilot experiment, see [Create an Amazon SageMaker Autopilot experiment \(p. 145\)](#)

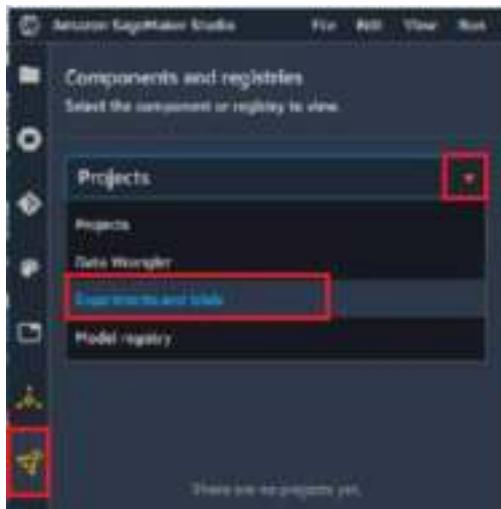
### Note

To access the feature importance metrics in this procedure, you must first select File > Shut down, and then restart Studio from the Console.

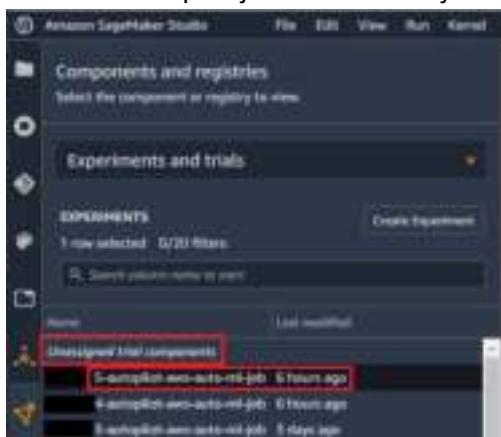
1. To view model details after running an Amazon SageMaker Autopilot job, select the triangular icon from the menu on the left to open the **Components and registries** page.



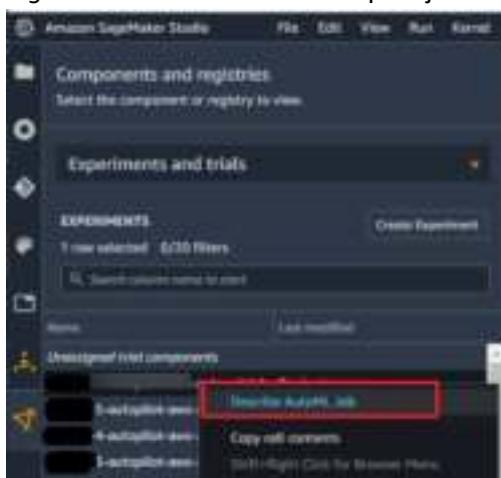
2. Select **Experiments and trials** from the dropdown menu.



3. Locate the Autopilot job whose details you want to examine in the **Unassigned trial components** list



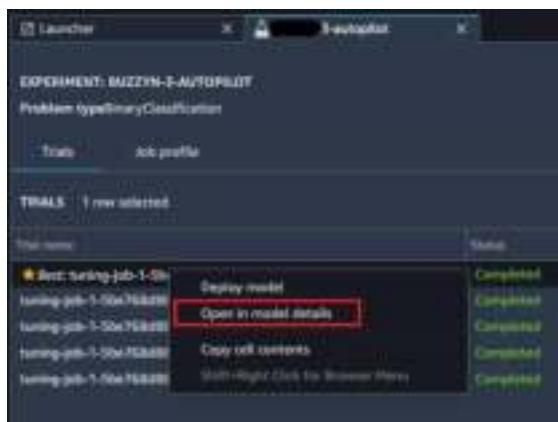
4. Right-click the name of the Autopilot job and select **Describe AutoML Job** from the pop up menu.



To review the Objective:F1 metrics for each tuning job.



5. To review the model details for the best tuning job, right-click the **Trial name** labeled as the **Best** at the top of the list of trials that has the highest objective metric score and select **Open Model Details**.



6. The plot of the aggregated SHAP values that indicate the importance of each feature is displayed in the **Explaining your model's predictions** section.



7. For more information about how the SHAP values help explain predictions based on feature importance, scroll down to see the link to the [Understanding the model explainability](#) whitepaper. Additional information is also available in the [Model Explainability \(p. 1852\)](#) topic in the SageMaker Developer Guide.



8. To see the summary statistics for the objective metric, the f1 values for training and validation, the training and validation errors, and a list of the parameter values used to train and tune the model, scroll down to the bottom of the page.



## Amazon SageMaker Autopilot notebooks generated to manage AutoML tasks

Amazon SageMaker Autopilot manages the key tasks in an automatic machine learning (AutoML) process. They are implemented by Autopilot with an AutoML job. The AutoML job creates two notebooks that describe the plan that Autopilot follows to generate candidate models. A candidate model consists of a (pipeline, algorithm) pair. First, there's a data exploration notebook, that describes what Autopilot learned about the data that you provided. Second, there's a candidate definition notebook, which uses the information about the data to generate candidates.

You can run both notebooks in Amazon SageMaker or locally if you have installed the [Amazon SageMaker Python SDK](#). You can share the notebooks just like any other SageMaker Studio notebook. The notebooks are created for you to conduct experiment. For example, you could edit the following items in the notebooks:

- the preprocessors used on the data
- the number of hyperparameter optimization (HPO) runs and their parallelism
- the algorithms to try
- the instance types used for the HPO jobs
- the hyperparameter ranges

Modifications to the candidate definition notebook are encouraged to be used as a learning tool. This capability allows you to learn about how the decisions made during the machine learning process impact the your results.

**Note**

When you run the notebooks in your default instance you incur baseline costs, but when you execute HPO jobs from the candidate notebook, these jobs use additional compute resources that incur additional costs.

## Data exploration notebook

A notebook is produced during the analysis phase of the AutoML job that helps you identify problems in your dataset. It identifies specific areas for investigation in order to help you identify upstream problems with your data that may result in a suboptimal model.

## Candidate definition notebook

The candidate definition notebook contains each suggested preprocessing step, algorithm, and hyperparameter ranges. If you chose just to produce the notebook and not to run the AutoML job, you can then decide which candidates are to be trained and tuned. They optimize automatically and a final, best candidate is identified. If you ran the job directly without seeing the candidates first, then only best candidate is displayed when you open the notebook after the completion of the job.

# Configure inference output in Autopilot-generated containers

Amazon SageMaker Autopilot generates an ordered [ContainerDefinition](#) list that can be used to build a model to deploy in a machine learning pipeline. This model can be used for online hosting and inference. Customers can access the list of inference container definitions with the [ListCandidateForAutoMLJob](#) API. The list of inference container definitions representing the best candidate is also available as part of the [DescribeAutoMLJob](#) response.

**Topics**

- [Inference container definitions for regression and classification problem types \(p. 157\)](#)
- [Select inference response for classification models \(p. 158\)](#)

## Inference container definitions for regression and classification problem types

The inference containers generated depend on the problem type of the job.

- **Regression:** generates two containers:
  1. First is the feature engineering container that transforms the original features to features that the regression algorithms can train on.
  2. Second is the algorithm container that transforms features and generates the regression score for the dataset.
- **Classification:** generates three containers:
  1. First is the feature engineering container that transforms the original features to features that the classification algorithms can train on.

2. Second is the algorithm container that generates the winning `predicted_label` and that can also produce the various probabilities associated with the classification outcomes in the inference response.
3. Third is a feature engineering container that performs post-processing of the algorithm prediction, for example, an inverse transform of the predicted label to original label.

## Select inference response for classification models

Classification inference containers allow you to select the content of the inference responses. There are four predefined keys:

- `predicted_label`: The winning label determined by Autopilot.
- `probability`: The probability of the `True` class for binary classification. The probability of winning class for multiclass classification.
- `probabilities`: The list of probabilities for all corresponding labels.
- `labels`: List of all labels

By default, inference containers are configured to generate `predicted_label` only.

Three environment variables are used to select the optional inference content:

- `SAGEMAKER_INFERENCE_SUPPORTED`: this is set to provide hints to the users about what content each container supports.
- `SAGEMAKER_INFERENCE_INPUT`: should be set to the keys that the container expects in its input payload.
- `SAGEMAKER_INFERENCE_OUTPUT`: should be populated with the set of keys the container outputs.

In order to choose the inference response content, we need to add the `SAGEMAKER_INFERENCE_INPUT`, `SAGEMAKER_INFERENCE_OUTPUT` appropriately in the second and the third containers in the list of containers for classification problem.

The keys supported by the third classification model container are `predicted_label`, `labels`, `probability` and `probabilities`. Hence the `SAGEMAKER_INFERENCE_SUPPORTED` environment includes the names of all these keys.

The keys supported by the second container (Algorithm) are `predicted_label`, `probability` and `probabilities`. Note that the `labels` is deliberately not added to the `SAGEMAKER_INFERENCE_SUPPORTED`.

Here is how to update the definition of the inference containers to receive `predicted_label` and `probability`.

```
containers[1]['Environment'].update({'SAGEMAKER_INFERENCE_OUTPUT': 'predicted_label,
probability'})
containers[2]['Environment'].update({'SAGEMAKER_INFERENCE_INPUT': 'predicted_label,
probability'})
containers[2]['Environment'].update({'SAGEMAKER_INFERENCE_OUTPUT': 'predicted_label,
probability'})
```

Here is how to update the definition of the inference containers to receive `predicted_label` and `probabilities` and `labels`. Note that you do not need to pass the `labels` to the second container, the algorithm container, as it's redundant and can be generated by the third container independently. This reduces the latency.

```
containers[1]['Environment'].update({'SAGEMAKER_INFERENCE_OUTPUT':
    'predicted_label,probabilities'})
containers[2]['Environment'].update({'SAGEMAKER_INFERENCE_INPUT':
    'predicted_label,probabilities'})
containers[2]['Environment'].update({'SAGEMAKER_INFERENCE_OUTPUT': 'predicted_label,
    probabilities,labels'})
```

You can use the [Amazon SageMaker Python SDK](#) to accomplish this as follows:

```
from sagemaker import AutoML

aml = AutoML.attach(auto_ml_job_name='AUTOML_JOB_NAME')
aml_best_model = aml.create_model(name='SELECT_MODEL_NAME',
                                    candidate=None,
                                    inference_response_keys=['probabilities', 'labels'])

aml_transformer = aml_best_model.transformer(accept='text/csv',
                                            assemble_with='Line',
                                            instance_type='ml.m5.xlarge',
                                            instance_count=1,)

aml_transformer.transform(test_data_s3_path,
                        content_type='text/csv',
                        split_type='Line',
                        job_name=<Add jobName>,
                        wait=True)
```

## Amazon SageMaker Autopilot quotas

There are quotas that limit the resources available to you when using Amazon SageMaker Autopilot. Some of these limits are increaseable and some are not.

### Topics

- [Quotas that you can increase \(p. 159\)](#)
- [Resource quotas \(p. 160\)](#)

## Quotas that you can increase

There are default limits for the size of datasets for the number of concurrent jobs you can run with Amazon SageMaker Autopilot for each Amazon account, per Amazon Region. You can increase these limits by contacting Amazon Support.

### Resource limits

Resource	Region	Default limits	Can be increased up to
Size of input dataset	All	5 GB	Hundreds of GBs
Number of concurrent Autopilot jobs	us-east-1, us-east-2, us-west-2, ap-northeast-1, eu-west-1, eu-central-1	4	Hundreds
	ap-northeast-2, ap-southeast-2, eu-west-2, ap-southeast-1	2	Hundreds

Resource	Region	Default limits	Can be increased up to
	All other regions	1	Tens

**To request a quota increase:**

1. Open the [Amazon Support Center](#) page, sign in if necessary, and then choose **Create case**.
2. On the **Create case** page, choose **Service limit increase**.
3. In the **Case details** panel, select **SageMaker AutoML** for the **Limit Type**.
4. On the **Requests** panel for **Request 1**, select the **Region**, the resource **Limit** to increase and the **New Limit** value you are requesting. Select **Add another request** if you have additional requests for quota increases.

The screenshot shows the 'Create case' interface. The 'Service limit increase' section is highlighted with a red box. In the 'Case details' panel, 'SageMaker AutoML' is selected as the limit type. The 'Requests' panel shows a note about requesting additional limit increases for different limit types. Request 1 is configured with the 'All-Join-Eligible Region' and '30 days' as the new limit.

5. Provide your preferred **Contact options** and choose **Submit**.

## Resource quotas

The following table contains the runtime resource limits for an Amazon SageMaker Autopilot job in an Amazon region.

### Resource limits per Autopilot job

Resource	Limit per Autopilot job
Maximum runtime for an Autopilot job	30 days

# API reference guide for Amazon SageMaker Autopilot

Amazon SageMaker provides API reference documentation that describes all of the REST operations and data types used by Autopilot and a higher level [Amazon SageMaker Python SDK](#) that you can use to create and manage AutoML jobs. It also provides a command line interface (CLI), an Amazon SDK for Python (Boto) for low-level clients of SageMaker services, and SDKs for .NET, C++, Go, Java, JavaScript, PHP V3, and Ruby V3. The following sections describe these Autopilot programming interfaces.

## Topics

- [SageMaker API reference \(p. 161\)](#)
- [Amazon SageMaker Python SDK \(p. 162\)](#)
- [Amazon Command Line Interface \(CLI\) \(p. 162\)](#)
- [Amazon SDK for Python \(Boto\) \(p. 162\)](#)
- [Amazon SDK for .NET \(p. 163\)](#)
- [Amazon SDK for C++ \(p. 163\)](#)
- [Amazon SDK for Go \(p. 163\)](#)
- [Amazon SDK for Java \(p. 163\)](#)
- [Amazon SDK for JavaScript \(p. 163\)](#)
- [Amazon SDK for PHP V3 \(p. 164\)](#)
- [Amazon SDK for Ruby V3 \(p. 164\)](#)

## SageMaker API reference

This API provides HTTP service APIs for creating and managing Amazon SageMaker Autopilot resources.

### Actions

- [CreateAutoMLJob](#)
- [DescribeAutoMLJob](#)
- [ListAutoMLJobs](#)
- [ListCandidatesForAutoMLJob](#)
- [StopAutoMLJob](#)

### Data Types

- [AutoMLCandidate](#)
- [AutoMLCandidateStep](#)
- [AutoMLChannel](#)
- [AutoMLContainerDefinition](#)
- [AutoMLDataSource](#)
- [AutoMLJobArtifacts](#)
- [AutoMLJobCompletionCriteria](#)
- [AutoMLJobConfig](#)
- [AutoMLJobObjective](#)
- [AutoMLJobSummary](#)

- [AutoMLOutputDataConfig](#)
- [AutoMLPartialFailureReason](#)
- [AutoMLS3DataSource](#)
- [AutoMLSecurityConfig](#)
- [CandidateArtifactLocations](#)
- [CandidateProperties](#)
- [ResolvedAttributes](#)

For more information on the entire SageMaker REST API, see [API and SDK Reference](#).

## Amazon SageMaker Python SDK

This Python library provides several high-level abstractions for working with SageMaker. The following classes can be used to manage AutoML jobs.

- [AutoML](#)
- [AutoMLInput](#)
- [AutoMLJob](#)
- [CandidateEstimator](#)
- [CandidateStep](#)

For more information how this Python SDK simplifies model training and deployment, see [Using the Amazon SageMaker Python SDK](#).

## Amazon Command Line Interface (CLI)

The [Amazon CLI](#) provides APIs for creating and managing SageMaker resources. Here are the [Amazon CLI for Amazon SageMaker](#) Autopilot commands.

- [create-auto-ml-job](#)
- [describe-auto-ml-job](#)
- [list-auto-ml-jobs](#)
- [list-candidates-for-auto-ml-job](#)
- [stop-auto-ml-job](#)

## Amazon SDK for Python (Boto)

Boto is the Amazon Web Services (Amazon) SDK for Python. It enables Python developers to create, configure, and manage Amazon services such as SageMaker. Boto provides a low-level [Client API](#) that maps to the underlying SageMaker service API. Here is a list of the methods used to manage AutoML jobs with the Client class.

- [create\\_auto\\_ml\\_job\(\)](#)
- [describe\\_auto\\_ml\\_job\(\)](#)
- [list\\_auto\\_ml\\_jobs\(\)](#)
- [list\\_candidates\\_for\\_auto\\_ml\\_job\(\)](#)
- [stop\\_auto\\_ml\\_job\(\)](#)

## Amazon SDK for .NET

The .NET SDK enables developers to create, configure, and manage Amazon services such as SageMaker. The API maps to the underlying SageMaker service API. Here is a list of the methods used to manage AutoML jobs with the Client class.

- [CreateAutoMLJob](#)
- [DescribeAutoMLJob](#)
- [ListAutoMLJobs](#)
- [ListCandidatesForAutoMLJob](#)
- [StopAutoMLJob](#)

## Amazon SDK for C++

The C++ SDK enables developers to create, configure, and manage Amazon services such as SageMaker. The API maps to the underlying SageMaker service API. For information on the methods used to manage AutoML jobs with the Client class, see [Aws::SageMaker::SageMakerClient Class Reference](#).

## Amazon SDK for Go

The Go SDK enables developers to create, configure, and manage Amazon services such as SageMaker. The API maps to the underlying SageMaker service API. Here is a list of the methods used to manage AutoML jobs with the Client class.

- [CreateAutoMLJob](#)
- [DescribeAutoMLJob](#)
- [ListAutoMLJobs](#)
- [ListCandidatesForAutoMLJob](#)
- [StopAutoMLJob](#)

## Amazon SDK for Java

The Java SDK enables developers to create, configure, and manage Amazon services such as SageMaker. The API maps to the underlying SageMaker service API. Here is a list of the methods used to manage AutoML jobs with the Client class.

- [createAutoMLJob](#)
- [describeAutoMLJob](#)
- [listAutoMLJobs](#)
- [listCandidatesForAutoMLJob](#)
- [stopAutoMLJob](#)

## Amazon SDK for JavaScript

The JavaScript SDK enables developers to create, configure, and manage Amazon services such as SageMaker. The API maps to the underlying SageMaker service API. Here is a list of the methods used to manage AutoML jobs with the Client class.

- [createAutoMLJob](#)

- [describeAutoMLJob](#)
- [listAutoMLJobs](#)
- [listCandidatesForAutoMLJob](#)
- [stopAutoMLJob](#)

## Amazon SDK for PHP V3

The PHP V3 SDK enables developers to create, configure, and manage Amazon services such as SageMaker. The API maps to the underlying SageMaker service API. Here is a list of the methods used to manage AutoML jobs with the Client class.

- [CreateAutoMLJob](#)
- [DescribeAutoMLJob](#)
- [ListAutoMLJobs](#)
- [ListCandidatesForAutoMLJob](#)
- [StopAutoMLJob](#)

## Amazon SDK for Ruby V3

The Ruby V3 SDK enables developers to create, configure, and manage Amazon services such as SageMaker. The API maps to the underlying SageMaker service API. Here is a list of the methods used to manage AutoML jobs with the Client class.

- [create\\_auto\\_ml\\_job\(\)](#)
- [describe\\_auto\\_ml\\_job\(\)](#)
- [list\\_auto\\_ml\\_jobs\(\)](#)
- [list\\_candidates\\_for\\_auto\\_ml\\_job\(\)](#)
- [stop\\_auto\\_ml\\_job\(\)](#)

# Label Data

These features are not available in the China Regions.

To train a machine learning model, you need a large, high-quality, labeled dataset. You can label your data using Amazon SageMaker Ground Truth. Choose from one of the Ground Truth [built-in task types](#) or create your own [custom labeling workflow](#). To improve the accuracy of your data labels and reduce the total cost of labeling your data, use Ground Truth enhanced data labeling features like [automated data labeling](#) and [annotation consolidation](#).

## Topics

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Create and Manage Workforces \(p. 456\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## Use Amazon SageMaker Ground Truth to Label Data

This feature is not available in the China Regions.

To train a machine learning model, you need a large, high-quality, labeled dataset. Ground Truth helps you build high-quality training datasets for your machine learning models. With Ground Truth, you can use workers from either Amazon Mechanical Turk, a vendor company that you choose, or an internal, private workforce along with machine learning to enable you to create a labeled dataset. You can use the labeled dataset output from Ground Truth to train your own models. You can also use the output as a training dataset for an Amazon SageMaker model.

Depending on your ML application, you can choose from one of the Ground Truth built-in task types to have workers generate specific types of labels for your data. You can also build a custom labeling workflow to provide your own UI and tools to workers labeling your data. To learn more about the Ground Truth built in task types, see [Built-in Task Types \(p. 333\)](#). To learn how to create a custom labeling workflow, see [Creating Custom Labeling Workflows \(p. 300\)](#).

In order to automate labeling your training dataset, you can optionally use [automated data labeling](#), a Ground Truth process that uses machine learning to decide which data needs to be labeled by humans. Automated data labeling may reduce the labeling time and manual effort required. For more information, see [Automate Data Labeling \(p. 430\)](#). To create a custom labeling workflow, see [Creating Custom Labeling Workflows \(p. 300\)](#).

Use either pre-built or custom tools to assign the labeling tasks for your training dataset. A *labeling UI template* is a webpage that Ground Truth uses to present tasks and instructions to your workers. The SageMaker console provides built-in templates for labeling data. You can use these templates to get

started , or you can build your own tasks and instructions by using our HTML 2.0 components. For more information, see [Creating Custom Labeling Workflows \(p. 300\)](#).

Use the workforce of your choice to label your dataset. You can choose your workforce from:

- The Amazon Mechanical Turk workforce of over 500,000 independent contractors worldwide.
- A private workforce that you create from your employees or contractors for handling data within your organization.
- A vendor company that you can find in the Amazon Web Services Marketplace that specializes in data labeling services.

For more information, see [Create and Manage Workforces \(p. 456\)](#).

You store your datasets in Amazon S3 buckets. The buckets contain three things: The data to be labeled, an input manifest file that Ground Truth uses to read the data files, and an output manifest file. The output file contains the results of the labeling job. For more information, see [Use Input and Output Data \(p. 362\)](#).

Events from your labeling jobs appear in Amazon CloudWatch under the /aws/sagemaker/LabelingJobs group. CloudWatch uses the labeling job name as the name for the log stream.

## Are You a First-time User of Ground Truth?

If you are a first-time user of Ground Truth, we recommend that you do the following:

1. **Read Getting started (p. 166)**—This section walks you through setting up your first Ground Truth labeling job.
2. **Explore other topics**—Depending on your needs, do the following:
  - **Explore built-in task types**— Use built-in task types to streamline the process of creating a labeling job. See [Built-in Task Types \(p. 333\)](#) to learn more about Ground Truth built-in task types.
  - **Manage your labeling workforce**—Create new work teams and manage your existing workforce. For more information, see [Create and Manage Workforces \(p. 456\)](#).
  - **Learn about streaming labeling jobs**— Create a streaming labeling job and send new dataset objects to workers in real time using a perpetually running labeling job. Workers continuously receive new data objects to label as long as the labeling job is active and new objects are being sent to it. To learn more, see [Ground Truth Streaming Labeling Jobs \(p. 366\)](#).
3. **See the Reference**—This section describes operations to automate Ground Truth operations.

## Getting started

This video shows you how to setup and use Amazon SageMaker Ground Truth. (Length: 9:37)

To get started using Amazon SageMaker Ground Truth, follow the instructions in the following sections. The sections here explain how to use the console to create a labeling job, assign a public or private workforce, and send the labeling job to your workforce. You can also learn how to monitor the progress of a labeling job.

If you want to create a custom labeling workflow, see [Creating Custom Labeling Workflows \(p. 300\)](#) for instructions.

Before you create a labeling job, you must upload your dataset to an Amazon S3 bucket. For more information, see [Use Input and Output Data \(p. 362\)](#).

### Topics

- [Step 1: Before You Begin \(p. 167\)](#)

- [Step 2: Create a Labeling Job \(p. 167\)](#)
- [Step 3: Select Workers \(p. 168\)](#)
- [Step 4: Configure the Bounding Box Tool \(p. 170\)](#)
- [Step 5: Monitoring Your Labeling Job \(p. 171\)](#)

## Step 1: Before You Begin

Before you begin using the SageMaker console to create a labeling job, you must set up the dataset for use. Do this:

1. Save two images at publicly available HTTP URLs. The images are used when creating instructions for completing a labeling task. The images should have an aspect ratio of around 2:1. For this exercise, the content of the images is not important.
2. Create an Amazon S3 bucket to hold the input and output files. The bucket must be in the same Region where you are running Ground Truth. Make a note of the bucket name because you use it during step 2.

Ground Truth requires all S3 buckets that contain labeling job input image data have a CORS policy attached. To learn more about this change, see [CORS Permission Requirement \(p. 439\)](#).

3. Assign the following permissions policy to the user that is creating the labeling job:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "sagemakergroundtruth",  
            "Effect": "Allow",  
            "Action": [  
                "cognito-idp:CreateGroup",  
                "cognito-idp:CreateUserPool",  
                "cognito-idp:CreateUserPoolDomain",  
                "cognito-idp:AdminCreateUser",  
                "cognito-idp:CreateUserPoolClient",  
                "cognito-idp:AdminAddUserToGroup",  
                "cognito-idp:DescribeUserPoolClient",  
                "cognito-idp:DescribeUserPool",  
                "cognito-idp:UpdateUserPool"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

## Next

[Step 2: Create a Labeling Job \(p. 167\)](#)

## Step 2: Create a Labeling Job

In this step you use the console to create a labeling job. You tell Amazon SageMaker Ground Truth the Amazon S3 bucket where the manifest file is stored and configure the parameters for the job. For more information about storing data in an Amazon S3 bucket, see [Use Input and Output Data \(p. 362\)](#).

### To create a labeling job

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.

2. From the left navigation, choose **Labeling jobs**.
3. Choose **Create labeling job** to start the job creation process.
4. In the **Job overview** section, provide the following information:
  - **Job name** – Give the labeling job a name that describes the job. This name is shown in your job list. The name must be unique in your account in an Amazon Region.
  - **Label attribute name** – Leave this unchecked as the default value is the best option for this introductory job.
  - **Input data setup** – Select **Automated data setup**. This option allows you to automatically connect to your input data in S3.
  - **S3 location for input datasets** – Enter the S3 location where you added the images in step 1.
  - **S3 location for output datasets** – The location where your output data is written in S3.
  - **Data type** – Use the drop down menu to select **Image**. Ground Truth will use all images found in the S3 location for input datasets as input for your labeling job.
  - **IAM role** – Create or choose an IAM role with the SageMakerFullAccess IAM policy attached.
5. In the **Task type** section, for the **Task category** field, choose **Image**.
6. In the **Task selection** choose **Bounding box**.
7. Choose **Next** to move on to configuring your labeling job.

## Next

[Step 3: Select Workers \(p. 168\)](#)

## Step 3: Select Workers

In this step you choose a workforce for labeling your dataset. It is recommended that you create a private workforce to test Amazon SageMaker Ground Truth. Use email addresses to invite the members of your workforce. If you create a private workforce in this step you won't be able to import your Amazon Cognito user pool later. If you want to create a private workforce using an Amazon Cognito user pool, see [Manage a Private Workforce \(Amazon Cognito\) \(p. 465\)](#) and use the Mechanical Turk workforce instead in this tutorial.

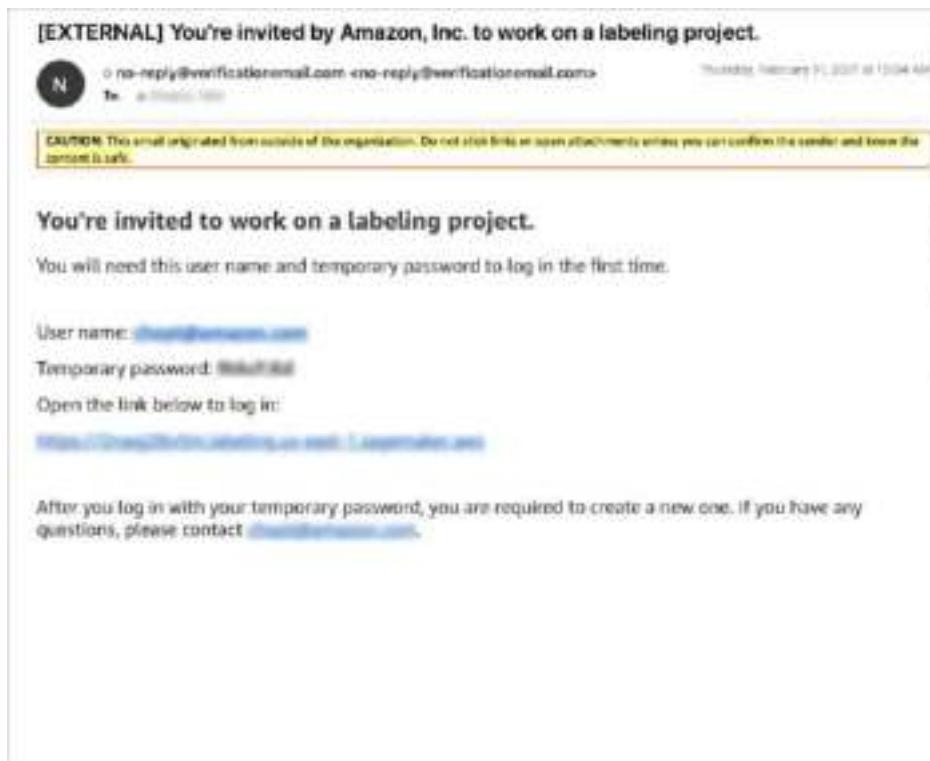
### Tip

To learn about the other workforce options you can use with Ground Truth, see [Create and Manage Workforces \(p. 456\)](#).

### To create a private workforce:

1. In the **Workers** section, choose **Private**.
2. If this is your first time using a private workforce, in the **Email addresses** field, enter up to 100 email addresses. The addresses must be separated by a comma. You should include your own email address so that you are part of the workforce and can see data object labeling tasks.
3. In the **Organization name** field, enter the name of your organization. This information is used to customize the email sent to invite a person to your private workforce.
4. In the **Contact email** field enter an email address that members of the workforce use to report problems with the task.

If you add yourself to the private workforce, you will receive an email that looks similar to the following. **Amazon, Inc.** is replaced by the organization you enter in step 3 of the preceding procedure. Select the link in the email to log in using the temporary password provided. If prompted, change your password. When you successfully log in, you see the worker portal where your labeling tasks appear.



**Tip**

You can find the link to your private workforce's worker portal in the **Labeling workforces** section of the Ground Truth area of the SageMaker console. To see the link, select the **Private** tab. The link is under the **Labeling portal sign-in URL** header in **Private workforce summary**.

If you choose to use the Amazon Mechanical Turk workforce to label the dataset, you are charged for labeling tasks completed on the dataset.

**To use the Amazon Mechanical Turk workforce:**

1. In the **Workers** section, choose **Public**.
2. Set a **Price per task**.
3. If applicable, choose **The dataset does not contain adult content** to acknowledge that the sample dataset has no adult content. This information enables Amazon SageMaker Ground Truth to warn external workers on Mechanical Turk that they might encounter potentially offensive content in your dataset.
4. Choose the check box next to the following statement to acknowledge that the sample dataset does not contain any personally identifiable information (PII). This is a requirement to use Mechanical Turk with Ground Truth. If your input data does contain PII, use the private workforce for this tutorial.

**You understand and agree that the Amazon Mechanical Turk workforce consists of independent contractors located worldwide and that you should not share confidential information, personal information or protected health information with this workforce.**

**Next**

[Step 4: Configure the Bounding Box Tool \(p. 170\)](#)

## Step 4: Configure the Bounding Box Tool

Finally you configure the bounding box tool to give instructions to your workers. You can configure a task title that describes the task and provides high-level instructions for the workers. You can provide both quick instructions and full instructions. Quick instructions are displayed next to the image to be labeled. Full instructions contain detailed instructions for completing the task. In this example, you only provide quick instructions. You can see an example of full instructions by choosing **Full instructions** at the bottom of the section.

### To configure the bounding box tool

1. In the **Task description** field type in brief instructions for the task. For example:

**Draw a box around any *objects* in the image.**

Replace *objects* with the name of an object that appears in your images.

2. In the **Labels** field, type a category name for the objects that the worker should draw a bounding box around. For example, if you are asking the worker to draw boxes around football players, you could use "Football Player" in this field.
3. The **Short instructions** section enables you to create instructions that are displayed on the page with the image that your workers are labeling. We suggest that you include an example of a correctly drawn bounding box and an example of an incorrectly drawn box. To create your own instructions, use these steps:
  - a. Select the text between **GOOD EXAMPLE** and the image placeholder. Replace it with the following text:

**Draw the box around the object with a small border.**
  - b. Select the first image placeholder and delete it.
  - c. Choose the image button and then enter the HTTPS URL of one of the images that you created in step 1. It is also possible to embed images directly in the short instructions section, however this section has a quota of 100 kilobytes (including text). If your images and text exceed 100 kilobytes, you receive an error.
  - d. Select the text between **BAD EXAMPLE** and the image placeholder. Replace it with the following text:

**Don't make the bounding box too large or cut into the object.**
  - e. Select the second image placeholder and delete it.
  - f. Choose the image button and then enter the HTTPS URL of the other image that you created in step 1.
4. Select **Preview** to preview the worker UI. The preview opens in a new tab, and so if your browser blocks pop ups you may need to manually enable the tab to open. When you add one or more annotations to the preview and then select **Submit** you can see a preview of the output data your annotation would create.
5. After you have configured and verified your instructions, select **Create** to create the labeling job.

If you used a private workforce, you can navigate to the worker portal that you logged into in [Step 3: Select Workers \(p. 168\)](#) of this tutorial to see your labeling tasks. The tasks may take a few minutes to appear.

[Next](#)

[Step 5: Monitoring Your Labeling Job \(p. 171\)](#)

## Step 5: Monitoring Your Labeling Job

After you create your labeling job, you see a list of all the jobs that you have created. You can use this list to monitor the status of your labeling jobs. The list has the following fields:

- **Name** – The name that you assigned the job when you created it.
- **Status** – The completion status of the job. The status can be one of Complete, Failed, In progress, or Stopped.
- **Labeled objects/total** – Shows the total number of objects in the labeling job and how many of them have been labeled.
- **Creation time** – The date and time that you created the job.

You can also clone, chain, or stop a job. Select a job and then select one of the following from the **Actions** menu:

- **Clone** – Creates a new labeling job with the configuration copied from the selected job. You can clone a job when you want to change to the job and run it again. For example, you can clone a job that was sent to a private workforce so that you can send it to the Amazon Mechanical Turk workforce. Or you can clone a job to rerun it against a new dataset stored in the same location as the original job.
- **Chain** – Creates a new labeling job that can build upon the data and models (if any) of a stopped, failed, or completed job. For more information about the use cases and how to use it, see [Chaining Labeling Jobs \(p. 436\)](#).
- **Stop** – Stops a running job. You cannot restart a stopped job. You can clone a job to start over or chain the job to continue from where it left off. Labels for any already labeled objects are written to the output file location. For more information, see [Output Data \(p. 404\)](#).

## Label Images

Use Ground Truth to label images. Select one of the following built-in task types to learn more about that task type. Each page includes instructions to help you create a labeling job using that task type.

### Tip

To learn more about supported file types and input data quotas, see [Input Data \(p. 363\)](#).

### Topics

- [Bounding Box \(p. 171\)](#)
- [Image Semantic Segmentation \(p. 177\)](#)
- [Image Classification \(Single Label\) \(p. 180\)](#)
- [Image Classification \(Multi-label\) \(p. 183\)](#)
- [Image Label Verification \(p. 187\)](#)

## Bounding Box

The images used to train a machine learning model often contain more than one object. To classify and localize one or more objects within images, use the Amazon SageMaker Ground Truth bounding box labeling job task type. In this context, localization means the pixel-location of the bounding box.

You create a bounding box labeling job using the Ground Truth section of the Amazon SageMaker console or the [CreateLabelingJob](#) operation.

**Important**

For this task type, if you create your own manifest file, use "source-ref" to identify the location of each image file in Amazon S3 that you want labeled. For more information, see [Input Data \(p. 363\)](#).

## Creating a Bounding Box Labeling Job (Console)

You can follow the instructions [Create a Labeling Job \(Console\) \(p. 336\)](#) to learn how to create a bounding box labeling job in the SageMaker console. In Step 10, choose **Image** from the **Task category** drop down menu, and choose **Bounding box** as the task type.

Ground Truth provides a worker UI similar to the following for labeling tasks. When you create the labeling job with the console, you specify instructions to help workers complete the job and up to 50 labels that workers can choose from.

Instructions

Shortcuts

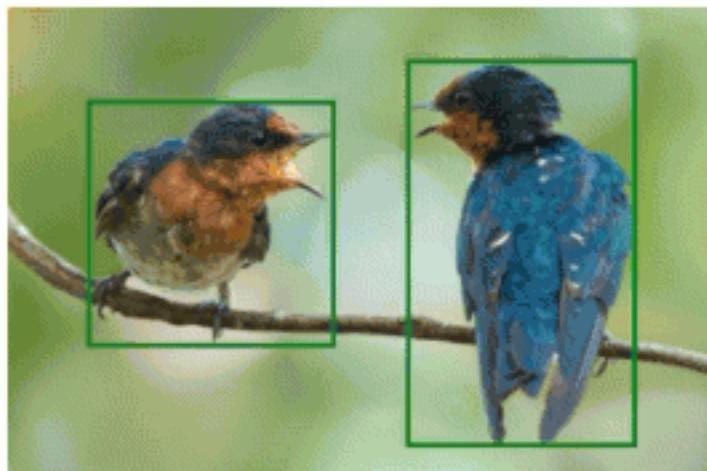
Select a category from the list and draw a box around the object.

Instructions

X

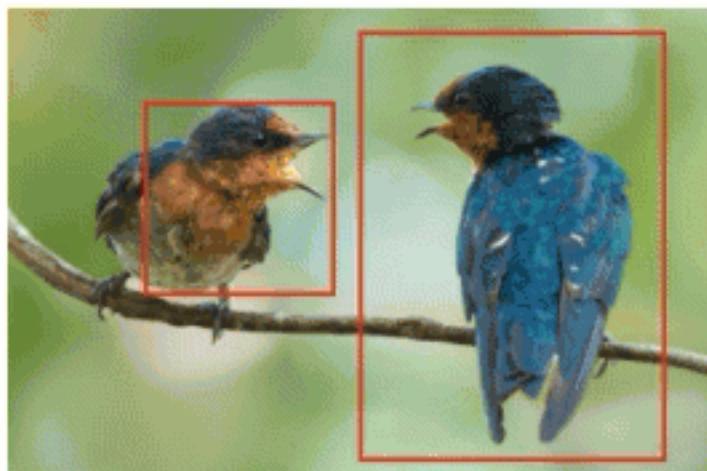
### Good example

Fit each box tightly around the boundaries of the object.



### Bad example

Boxes should not overlap with the boundaries of objects.



## Create a Bounding Box Labeling Job (API)

To create a bounding box labeling job, use the SageMaker API operation `CreateLabelingJob`. This API defines this operation for all Amazon SDKs. To see a list of language-specific SDKs supported for this operation, review the **See Also** section of [CreateLabelingJob](#).

Follow the instructions on [Create a Labeling Job \(API\) \(p. 338\)](#) and do the following while you configure your request:

- Pre-annotation Lambda functions for this task type end with `PRE-BoundingBox`. To find the pre-annotation Lambda ARN for your Region, see [PreHumanTaskLambdaArn](#).
- Annotation-consolidation Lambda functions for this task type end with `ACS-BoundingBox`. To find the annotation-consolidation Lambda ARN for your Region, see [AnnotationConsolidationLambdaArn](#).

The following is an example of an [Amazon Python SDK \(Boto3\) request](#) to create a labeling job in the US East (N. Virginia) Region. All parameters in red should be replaced with your specifications and resources.

```
response = client.create_labeling_job(
    LabelingJobName='example-bounding-box-labeling-job',
    LabelAttributeName='label',
    InputConfig={
        'DataSource': {
            'S3DataSource': {
                'ManifestS3Uri': 's3://bucket/path/manifest-with-input-data.json'
            }
        },
        'DataAttributes': {
            'ContentClassifiers': [
                'FreeOfPersonallyIdentifiableInformation' | 'FreeOfAdultContent',
            ]
        }
    },
    OutputConfig={
        'S3OutputPath': 's3://bucket/path/file-to-store-output-data',
        'KmsKeyId': 'string'
    },
    RoleArn='arn:aws:iam::*:role/*',
    LabelCategoryConfigS3Uri='s3://bucket/path/label-categories.json',
    StoppingConditions={
        'MaxHumanLabeledObjectCount': 123,
        'MaxPercentageOfInputDatasetLabeled': 123
    },
    HumanTaskConfig={
        'WorkteamArn': 'arn:aws:sagemaker:region::workteam/private-crowd/*',
        'UiConfig': {
            'UiTemplateS3Uri': 's3://bucket/path/worker-task-template.html'
        },
        'PreHumanTaskLambdaArn': 'arn:aws:lambda:us-east-1:432418664414:function:PRE-
BoundingBox',
        'TaskKeywords': [
            'Bounding Box',
        ],
        'TaskTitle': 'Bounding Box task',
        'TaskDescription': 'Draw bounding boxes around objects in an image',
        'NumberOfHumanWorkersPerDataObject': 123,
        'TaskTimeLimitInSeconds': 123,
        'TaskAvailabilityLifetimeInSeconds': 123,
        'MaxConcurrentTaskCount': 123,
        'AnnotationConsolidationConfig': {
            'AnnotationConsolidationLambdaArn': 'arn:aws:lambda:us-
east-1:432418664414:function:ACS-BoundingBox'
        },
    }
)
```

```

    Tags=[  
      {  
        'Key': 'string',  
        'Value': 'string'  
      },  
    ]  

)

```

### Provide a Template for Bounding Box Labeling Jobs

If you create a labeling job using the API, you must supply a worker task template in `UiTemplateS3Uri`. Copy and modify the following template. Only modify the `short-instructions`, `full-instructions`, and `header`. Upload this template to S3, and provide the S3 URI for this file in `UiTemplateS3Uri`.

```

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>  
<crowd-form>  
  <crowd-bounding-box  
    name="boundingBox"  
    src="{{ task.input.taskObject | grant_read_access }}"  
    header="please draw box"  
    labels="{{ task.input.labels | to_json | escape }}"  
  >  
  
  <full-instructions header="Bounding box instructions">  
    <ol><li><strong>Inspect</strong> the image</li><li><strong>Determine</strong>  
    if the specified label is/are visible in the picture.</li>  
    <li><strong>Outline</strong> each instance of the specified label in the image using  
    the provided "Box" tool.</li></ol>  
    <ul><li>Boxes should fit tight around each object</li>  
    <li>Do not include parts of the object are overlapping or that cannot be seen, even  
    though you think you can interpolate the whole shape.</li>  
    <li>Avoid including shadows.</li>  
    <li>If the target is off screen, draw the box up to the edge of the image.</li>  
  </full-instructions>  
  
  <short-instructions>  
    <h3><span style="color: rgb(0, 138, 0);>Good example</span></h3>  
    <p>Enter description of a correct bounding box label and add images</p>  
    <h3><span style="color: rgb(230, 0, 0);>Bad example</span></h3>  
    <p>Enter description of an incorrect bounding box label and add images</p>  
  </short-instructions>  
  
  </crowd-bounding-box>  
</crowd-form>

```

### Bounding Box Output Data

Once you have created a bounding box labeling job, your output data will be located in the Amazon S3 bucket specified in the `S3OutputPath` parameter when using the API or in the **Output dataset location** field of the **Job overview** section of the console.

For example, the output manifest file of a successfully completed single-class bounding box task will contain the following:

```

[  
  {  
    "boundingBox": {  
      "boundingBoxes": [  
        {  
          "height": 2832,

```

```

        "label": "bird",
        "left": 681,
        "top": 599,
        "width": 1364
    }
],
"inputImageProperties": {
    "height": 3726,
    "width": 2662
}
}
]

```

The `boundingBoxes` parameter identifies the location of the bounding box drawn around an object identified as a "bird" relative to the top-left corner of the image which is taken to be the (0,0) pixel-coordinate. In the previous example, `left` and `top` identify the location of the pixel in the top-left corner of the bounding box relative to the top-left corner of the image. The dimensions of the bounding box are identified with `height` and `width`. The `inputImageProperties` parameter gives the pixel-dimensions of the original input image.

When you use the bounding box task type, you can create single- and multi-class bounding box labeling jobs. The output manifest file of a successfully completed multi-class bounding box will contain the following:

```

[
{
    "boundingBox": {
        "boundingBoxes": [
            {
                "height": 938,
                "label": "squirrel",
                "left": 316,
                "top": 218,
                "width": 785
            },
            {
                "height": 825,
                "label": "rabbit",
                "left": 1930,
                "top": 2265,
                "width": 540
            },
            {
                "height": 1174,
                "label": "bird",
                "left": 748,
                "top": 2113,
                "width": 927
            },
            {
                "height": 893,
                "label": "bird",
                "left": 1333,
                "top": 847,
                "width": 736
            }
        ],
        "inputImageProperties": {
            "height": 3726,
            "width": 2662
        }
    }
]
```

```
    }
```

To learn more about the output manifest file that results from a bounding box labeling job, see [Bounding Box Job Output \(p. 409\)](#).

To learn more about the output manifest file generated by Ground Truth and the file structure the Ground Truth uses to store your output data, see [Output Data \(p. 404\)](#).

## Image Semantic Segmentation

To identify the contents of an image at the pixel level, use an Amazon SageMaker Ground Truth semantic segmentation labeling task. When assigned a semantic segmentation labeling job, workers classify pixels in the image into a set of predefined labels or classes. Ground Truth supports single and multi-class semantic segmentation labeling jobs.

You create a semantic segmentation labeling job using the Ground Truth section of the Amazon SageMaker console or the [CreateLabelingJob](#) operation.

### Important

For this task type, if you create your own manifest file, use "source-ref" to identify the location of each image file in Amazon S3 that you want labeled. For more information, see [Input Data \(p. 363\)](#).

### Creating a Semantic Segmentation Labeling Job (Console)

You can follow the instructions [Create a Labeling Job \(Console\) \(p. 336\)](#) to learn how to create a semantic segmentation labeling job in the SageMaker console. In Step 10, choose **Image** from the **Task category** drop down menu, and choose **Semantic segmentation** as the task type.

Ground Truth provides a worker UI similar to the following for labeling tasks. When you create the labeling job with the console, you specify instructions to help workers complete the job and labels that workers can choose from.

## Instructions

X

For each animal in the image, label all pixels that are part of the animal with the appropriate label color.

[View full instructions](#)

[View tool guide](#)

[How to use the Auto-segmentation tool](#)

### Good example

All pixels in the image that are part of an animal have been colored with the appropriate label color.

### Bad example

Some animals in the image have not been colored in completely.

The color for a given animal extends beyond the boundaries of the animal.

## Create a Semantic Segmentation Labeling Job (API)

To create a semantic segmentation labeling job, use the SageMaker API operation `CreateLabelingJob`. This API defines this operation for all Amazon SDKs. To see a list of language-specific SDKs supported for this operation, review the **See Also** section of [CreateLabelingJob](#).

Follow the instructions on [Create a Labeling Job \(API\) \(p. 338\)](#) and do the following while you configure your request:

- Pre-annotation Lambda functions for this task type end with `PRE-SemanticSegmentation`. To find the pre-annotation Lambda ARN for your Region, see [PreHumanTaskLambdaArn](#).
- Annotation-consolidation Lambda functions for this task type end with `ACS-SemanticSegmentation`. To find the annotation-consolidation Lambda ARN for your Region, see [AnnotationConsolidationLambdaArn](#).

The following is an example of an [Amazon Python SDK \(Boto3\) request](#) to create a labeling job in the US East (N. Virginia) Region. All parameters in red should be replaced with your specifications and resources.

```
response = client.create_labeling_job(
    LabelingJobName='example-semantic-segmentation-labeling-job',
    LabelAttributeName='label',
    InputConfig={
        'DataSource': {
            'S3DataSource': {
                'ManifestS3Uri': 's3://bucket/path/manifest-with-input-data.json'
            }
        },
        'DataAttributes': {
            'ContentClassifiers': [
                'FreeOfPersonallyIdentifiableInformation' | 'FreeOfAdultContent',
            ]
        }
    },
    OutputConfig={
        'S3OutputPath': 's3://bucket/path/file-to-store-output-data',
        'KmsKeyId': 'string'
    },
    RoleArn='arn:aws:iam::*:role/*',
    LabelCategoryConfigS3Uri='s3://bucket/path/label-categories.json',
    StoppingConditions={
        'MaxHumanLabeledObjectCount': 123,
        'MaxPercentageOfInputDatasetLabeled': 123
    },
    HumanTaskConfig={
        'WorkteamArn': 'arn:aws:sagemaker:region::workteam/private-crowd/*',
        'UiConfig': {
            'UiTemplateS3Uri': 's3://bucket/path/worker-task-template.html'
        },
        'PreHumanTaskLambdaArn': 'arn:aws:lambda:us-east-1:432418664414:function:PRE-SemanticSegmentation',
        'TaskKeywords': [
            'Semantic Segmentation',
        ],
        'TaskTitle': 'Semantic segmentation task',
        'TaskDescription': 'For each category provided, segment out each relevant object using the color associated with that category',
        'NumberOfHumanWorkersPerDataObject': 123,
        'TaskTimeLimitInSeconds': 123,
        'TaskAvailabilityLifetimeInSeconds': 123,
        'MaxConcurrentTaskCount': 123,
        'AnnotationConsolidationConfig': {
    }
```

```
'AnnotationConsolidationLambdaArn': 'arn:aws:lambda:us-
east-1:432418664414:function:ACS-SemanticSegmentation'
},
Tags:[
{
    'Key': 'string',
    'Value': 'string'
},
]
)
```

### Provide a Template for Semantic Segmentation Labeling Jobs

If you create a labeling job using the API, you must supply a worker task template in `UiTemplateS3Uri`. Copy and modify the following template. Only modify the `short-instructions`, `full-instructions`, and header.

Upload this template to S3, and provide the S3 URI for this file in `UiTemplateS3Uri`.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-semantic-segmentation
    name="crowd-semantic-segmentation"
    src="{{ task.input.taskObject | grant_read_access }}"
    header="Please segment out all pedestrians."
    labels="{{ task.input.labels | to_json | escape }}"
  >
    <full-instructions header="Segmentation instructions">
      <ol><li><strong>Read</strong> the task carefully and inspect the image.</li>
      <li><strong>Read</strong> the options and review the examples provided to understand more about the labels.</li>
      <li><strong>Choose</strong> the appropriate label that best suits an object and paint that object using the tools provided.</li>
    </ol>
    </full-instructions>
    <short-instructions>
      <h2><span style="color: rgb(0, 138, 0);>Good example</span></h2>
      <p>Enter description to explain a correctly done segmentation</p>
      <p><br></p><h2><span style="color: rgb(230, 0, 0);>Bad example</span></h2>
      <p>Enter description of an incorrectly done segmentation</p>
    </short-instructions>
  </crowd-semantic-segmentation>
</crowd-form>
```

### Semantic Segmentation Output Data

Once you have created a semantic segmentation labeling job, your output data will be located in the Amazon S3 bucket specified in the `S3OutputPath` parameter when using the API or in the **Output dataset location** field of the **Job overview** section of the console.

To learn more about the output manifest file generated by Ground Truth and the file structure the Ground Truth uses to store your output data, see [Output Data \(p. 404\)](#).

To see an example of an output manifest file for a semantic segmentation labeling job, see [3D Point Cloud Semantic Segmentation Output \(p. 420\)](#).

## Image Classification (Single Label)

Use an Amazon SageMaker Ground Truth image classification labeling task when you need workers to classify images using predefined labels that you specify. Workers are shown images and are asked to choose one label for each image.

You can create an image classification labeling job using the Ground Truth section of the Amazon SageMaker console or the [CreateLabelingJob](#) operation.

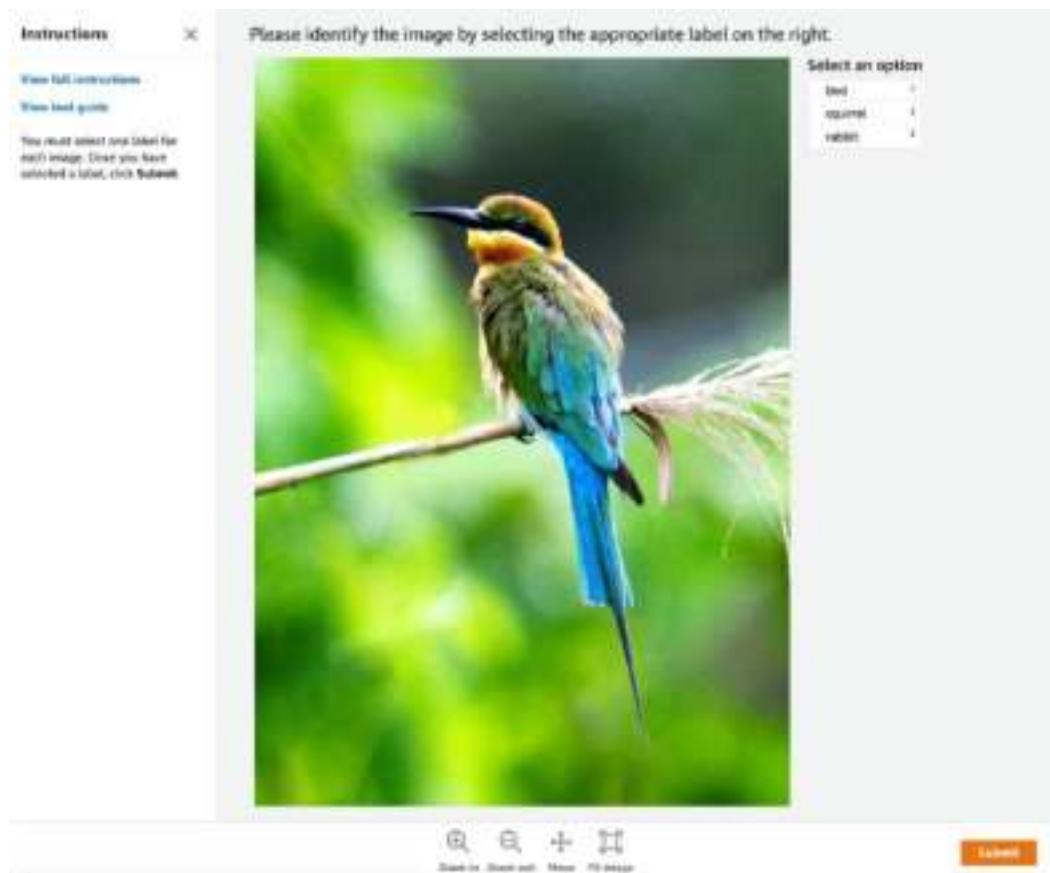
**Important**

For this task type, if you create your own manifest file, use "source-ref" to identify the location of each image file in Amazon S3 that you want labeled. For more information, see [Input Data \(p. 363\)](#).

## Create an Image Classification Labeling Job (Console)

You can follow the instructions [Create a Labeling Job \(Console\) \(p. 336\)](#) to learn how to create a image classification labeling job in the SageMaker console. In Step 10, choose **Image** from the **Task category** drop down menu, and choose **Image Classification (Single Label)** as the task type.

Ground Truth provides a worker UI similar to the following for labeling tasks. When you create the labeling job with the console, you specify instructions to help workers complete the job and labels that workers can choose from.



## Create an Image Classification Labeling Job (API)

To create an image classification labeling job, use the SageMaker API operation [CreateLabelingJob](#). This API defines this operation for all Amazon SDKs. To see a list of language-specific SDKs supported for this operation, review the **See Also** section of [CreateLabelingJob](#).

Follow the instructions on [Create a Labeling Job \(API\) \(p. 338\)](#) and do the following while you configure your request:

- Pre-annotation Lambda functions for this task type end with `PRE-ImageMultiClass`. To find the pre-annotation Lambda ARN for your Region, see [PreHumanTaskLambdaArn](#) .

- Annotation-consolidation Lambda functions for this task type end with ACS-ImageMultiClass. To find the annotation-consolidation Lambda ARN for your Region, see [AnnotationConsolidationLambdaArn](#).

The following is an example of an [Amazon Python SDK \(Boto3\) request](#) to create a labeling job in the US East (N. Virginia) Region. All parameters in red should be replaced with your specifications and resources.

```

response = client.create_labeling_job(
    LabelingJobName='example-image-classification-labeling-job',
    LabelAttributeName='label',
    InputConfig={
        'DataSource': {
            'S3DataSource': {
                'ManifestS3Uri': 's3://bucket/path/manifest-with-input-data.json'
            }
        },
        'DataAttributes': {
            'ContentClassifiers': [
                'FreeOfPersonallyIdentifiableInformation' | 'FreeOfAdultContent',
            ]
        }
    },
    OutputConfig={
        'S3OutputPath': 's3://bucket/path/file-to-store-output-data',
        'KmsKeyId': 'string'
    },
    RoleArn='arn:aws:iam::*:role/*',
    LabelCategoryConfigS3Uri='s3://bucket/path/label-categories.json',
    StoppingConditions={
        'MaxHumanLabeledObjectCount': 123,
        'MaxPercentageOfInputDatasetLabeled': 123
    },
    HumanTaskConfig={
        'WorkteamArn': 'arn:aws:sagemaker:region::workteam/private-crowd/*',
        'UiConfig': {
            'UiTemplateS3Uri': 's3://bucket/path/worker-task-template.html'
        },
        'PreHumanTaskLambdaArn': 'arn:aws:lambda:us-east-1:432418664414:function:PRE-
ImageMultiClass,
        'TaskKeywords': [
            'Image classification',
        ],
        'TaskTitle': 'Image classification task',
        'TaskDescription': 'Carefully inspect the image and classify it by selecting one
label from the categories provided.',
        'NumberOfHumanWorkersPerDataObject': 123,
        'TaskTimeLimitInSeconds': 123,
        'TaskAvailabilityLifetimeInSeconds': 123,
        'MaxConcurrentTaskCount': 123,
        'AnnotationConsolidationConfig': {
            'AnnotationConsolidationLambdaArn': 'arn:aws:lambda:us-
east-1:432418664414:function:ACS-ImageMultiClass'
        },
        Tags=[
            {
                'Key': 'string',
                'Value': 'string'
            },
        ],
    }
)

```

## Provide a Template for Image Classification Labeling Jobs

If you create a labeling job using the API, you must supply a worker task template in `UiTemplateS3Uri`. Copy and modify the following template. Only modify the `short-instructions`, `full-instructions`, and header.

Upload this template to S3, and provide the S3 URI for this file in `UiTemplateS3Uri`.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-image-classifier
    name="crowd-image-classifier"
    src="{{ task.input.taskObject | grant_read_access }}"
    header="please classify"
    categories="{{ task.input.labels | to_json | escape }}"
  >
    <full-instructions header="Image classification instructions">
      <ol><li><strong>Read</strong> the task carefully and inspect the image.</li>
      <li><strong>Read</strong> the options and review the examples provided to understand more about the labels.</li>
      <li><strong>Choose</strong> the appropriate label that best suits the image.</li></ol>
    </full-instructions>
    <short-instructions>
      <h3><span style="color: #008000;">Good example</span></h3>
      <p>Enter description to explain the correct label to the workers</p>
      <h3><span style="color: #C00000;">Bad example</span></h3><p>Enter description of an incorrect label</p>
    </short-instructions>
  </crowd-image-classifier>
</crowd-form>
```

## Image Classification Output Data

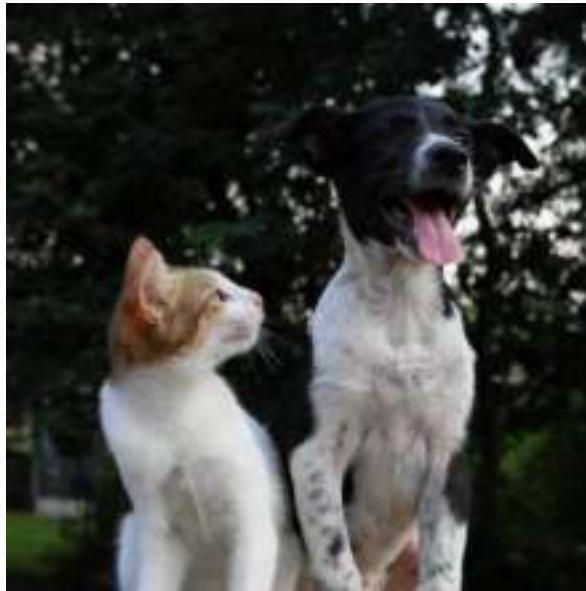
Once you have created an image classification labeling job, your output data will be located in the Amazon S3 bucket specified in the `S3OutputPath` parameter when using the API or in the **Output dataset location** field of the **Job overview** section of the console.

To learn more about the output manifest file generated by Ground Truth and the file structure the Ground Truth uses to store your output data, see [Output Data \(p. 404\)](#).

To see an example of an output manifest file from an image classification labeling job, see [Classification Job Output \(p. 408\)](#).

## Image Classification (Multi-label)

Use an Amazon SageMaker Ground Truth multi-label image classification labeling task when you need workers to classify multiple objects in an image. For example, the following image features a dog and a cat. You can use multi-label image classification to associate the labels "dog" and "cat" with this image.



When working on a multi-label image classification task, workers should choose all applicable labels, but must choose at least one. When creating a job using this task type, you can provide up to 50 label-categories.

When creating a labeling job in the console, Ground Truth doesn't provide a "none" category for when none of the labels applies to an image. To provide this option to workers, include a label similar to "none" or "other" when you create a multi-label image classification job.

To restrict workers to choosing a single label for each image, use the [Image Classification \(Single Label\) \(p. 180\)](#) task type.

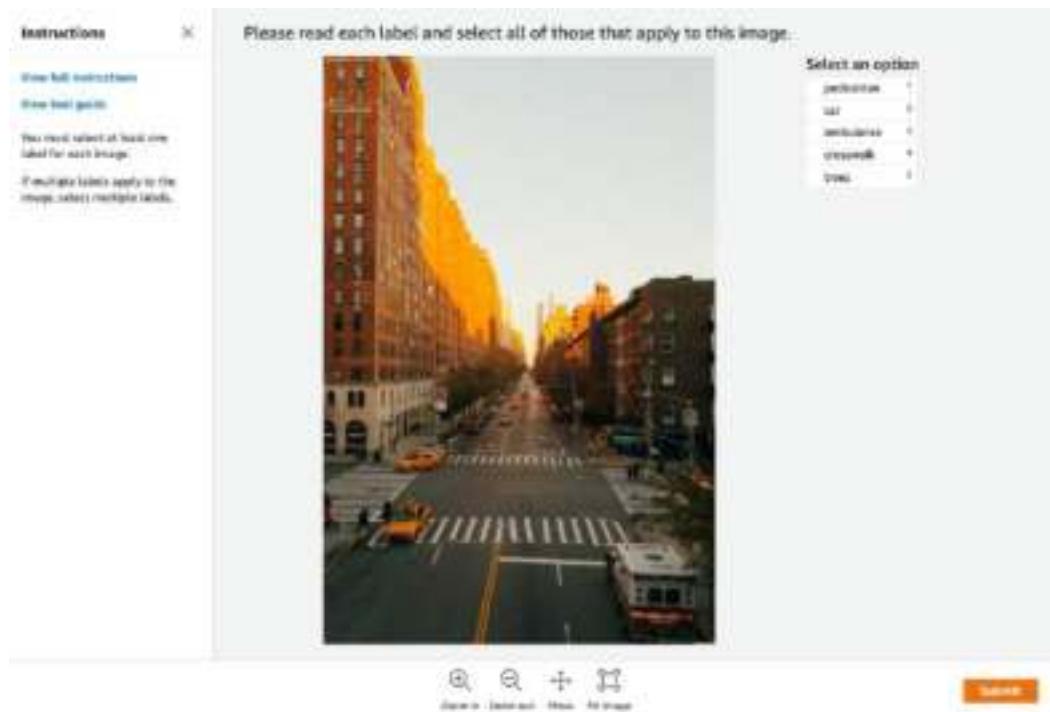
**Important**

For this task type, if you create your own manifest file, use "source-ref" to identify the location of each image file in Amazon S3 that you want labeled. For more information, see [Input Data \(p. 363\)](#).

## Create a Multi-Label Image Classification Labeling Job (Console)

You can follow the instructions [Create a Labeling Job \(Console\) \(p. 336\)](#) to learn how to create a multi-label image classification labeling job in the SageMaker console. In Step 10, choose **Image** from the **Task category** drop down menu, and choose **Image Classification (Multi-label)** as the task type.

Ground Truth provides a worker UI similar to the following for labeling tasks. When you create a labeling job in the console, you specify instructions to help workers complete the job and labels that workers can choose from.



## Create a Multi-Label Image Classification Labeling Job (API)

To create a multi-label image classification labeling job, use the SageMaker API operation `CreateLabelingJob`. This API defines this operation for all Amazon SDKs. To see a list of language-specific SDKs supported for this operation, review the **See Also** section of [CreateLabelingJob](#).

Follow the instructions on [Create a Labeling Job \(API\) \(p. 338\)](#) and do the following while you configure your request:

- Pre-annotation Lambda functions for this task type end with `PRE-ImageMultiClassMultiLabel`. To find the pre-annotation Lambda ARN for your Region, see [PreHumanTaskLambdaArn](#).
- Annotation-consolidation Lambda functions for this task type end with `ACS-ImageMultiClassMultiLabel`. To find the annotation-consolidation Lambda ARN for your Region, see [AnnotationConsolidationLambdaArn](#).

The following is an example of an [Amazon Python SDK \(Boto3\) request](#) to create a labeling job in the US East (N. Virginia) Region. All parameters in red should be replaced with your specifications and resources.

```
response = client.create_labeling_job(
    LabelingJobName='example-multi-label-image-classification-labeling-job',
    LabelAttributeName='label',
    InputConfig={
        'DataSource': {
            'S3DataSource': {
                'ManifestS3Uri': 's3://bucket/path/manifest-with-input-data.json'
            }
        },
        'DataAttributes': {
            'ContentClassifiers': [
                'FreeOfPersonallyIdentifiableInformation' | 'FreeOfAdultContent',
            ]
        }
    },
}
```

```

        OutputConfig={
            'S3OutputPath': 's3://bucket/path/file-to-store-output-data',
            'KmsKeyId': 'string'
        },
        RoleArn='arn:aws:iam::*:role/*',
        LabelCategoryConfigS3Uri='s3://bucket/path/label-categories.json',
        StoppingConditions={
            'MaxHumanLabeledObjectCount': 123,
            'MaxPercentageOfInputDatasetLabeled': 123
        },
        HumanTaskConfig={
            'WorkteamArn': 'arn:aws:sagemaker:region::workteam/private-crowd/*',
            'UiConfig': {
                'UiTemplateS3Uri': 's3://bucket/path/worker-task-template.html'
            },
            'PreHumanTaskLambdaArn': 'arn:aws:lambda:us-east-1:432418664414:function:PRE-
ImageMultiClassMultiLabel',
            'TaskKeywords': [
                'Image Classification',
            ],
            'TaskTitle': 'Multi-label image classification task',
            'TaskDescription': 'Select all labels that apply to the images shown',
            'NumberOfHumanWorkersPerDataObject': 123,
            'TaskTimeLimitInSeconds': 123,
            'TaskAvailabilityLifetimeInSeconds': 123,
            'MaxConcurrentTaskCount': 123,
            'AnnotationConsolidationConfig': {
                'AnnotationConsolidationLambdaArn': 'arn:aws:lambda:us-
east-1:432418664414:function:ACS-ImageMultiClassMultiLabel'
            },
            Tags=[
                {
                    'Key': 'string',
                    'Value': 'string'
                },
            ],
        }
    )
)

```

## Provide a Template for Multi-label Image Classification

If you create a labeling job using the API, you must supply a worker task template in `UiTemplateS3Uri`. Copy and modify the following template. Only modify the `short-instructions`, `full-instructions`, and header.

Upload this template to S3, and provide the S3 URI for this file in `UiTemplateS3Uri`.

```

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
    <crowd-image-classifier-multi-select
        name="crowd-image-classifier-multi-select"
        src="{{ task.input.taskObject | grant_read_access }}"
        header="Please identify all classes in image"
        categories="{{ task.input.labels | to_json | escape }}"
    >
        <full-instructions header="Multi Label Image classification instructions">
            <ol><li><strong>Read</strong> the task carefully and inspect the image.</li>
            <li><strong>Read</strong> the options and review the examples provided to understand
            more about the labels.</li>
            <li><strong>Choose</strong> the appropriate labels that best suit the image.</li>
        <ol>
            </full-instructions>
            <short-instructions>
                <h3><span style="color: rgb(0, 138, 0);>Good example</span></h3>
                <p>Enter description to explain the correct label to the workers</p>

```

```
<h3><span style="color: #c00000;">Bad example</span></h3>
<p>Enter description of an incorrect label</p>
</short-instructions>
</crowd-image-classifier-multi-select>
</crowd-form>
```

## Multi-label Image Classification Output Data

Once you have created a multi-label image classification labeling job, your output data will be located in the Amazon S3 bucket specified in the `S3OutputPath` parameter when using the API or in the **Output dataset location** field of the **Job overview** section of the console.

To learn more about the output manifest file generated by Ground Truth and the file structure the Ground Truth uses to store your output data, see [Output Data \(p. 404\)](#).

To see an example of output manifest files for multi-label image classification labeling job, see [Multi-label Classification Job Output \(p. 408\)](#).

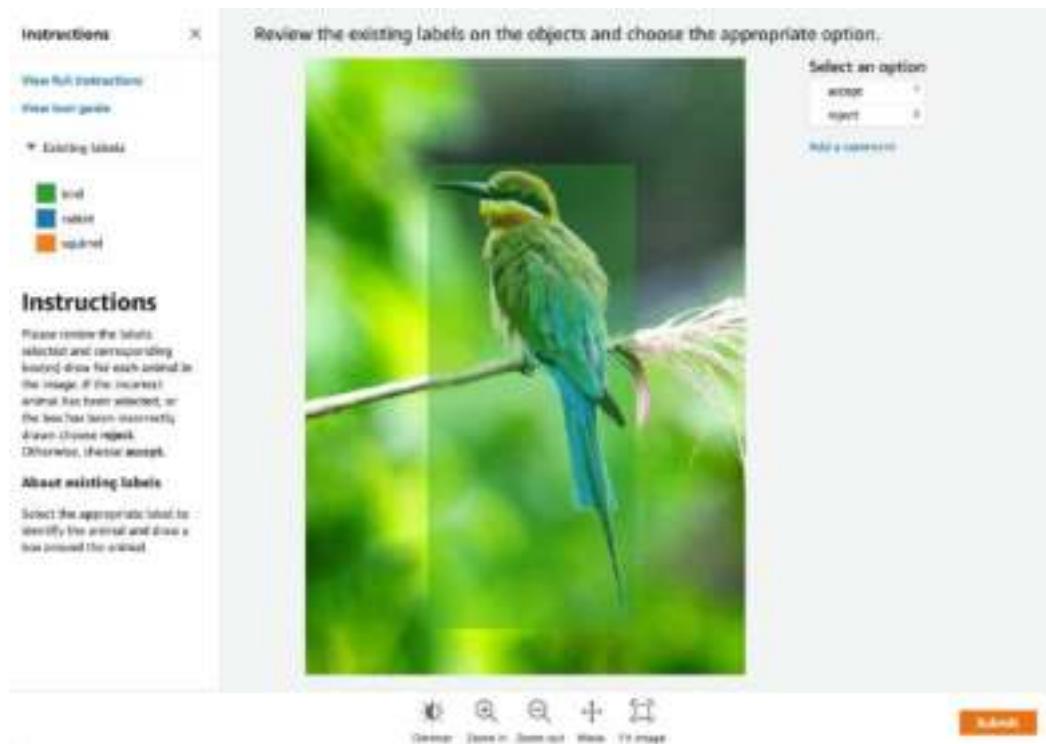
## Image Label Verification

Building a highly accurate training dataset for your machine learning (ML) algorithm is an iterative process. Typically, you review and continuously adjust your labels until you are satisfied that they accurately represent the ground truth, or what is directly observable in the real world.

You can use an Amazon SageMaker Ground Truth image label verification task to direct workers to review a dataset's labels and improve label accuracy. Workers can indicate if the existing labels are correct or rate label quality. They can also add comments to explain their reasoning. Amazon SageMaker Ground Truth supports label verification for [Bounding Box \(p. 171\)](#) and [Image Semantic Segmentation \(p. 177\)](#) labels.

You create an image label verification labeling job using the Ground Truth section of the Amazon SageMaker console or the [CreateLabelingJob](#) operation.

Ground Truth provides a worker console similar to the following for labeling tasks. When you create the labeling job with the console, you can modify the images and content that are shown. To learn how to create a labeling job using the Ground Truth console, see [Create a Labeling Job \(Console\) \(p. 336\)](#).



You can create a label verification labeling job using the SageMaker console or API. To learn how to create a labeling job using the Ground Truth API operation `CreateLabelingJob`, see [Create a Labeling Job \(API\) \(p. 338\)](#).

## Use Ground Truth to Label Text

Use Ground Truth to text. Select one of the following built in task types to learn more about that task type. Each page includes instructions to help you create a labeling job using that task type.

### Tip

To learn more about supported file types and input data quotas, see [Input Data \(p. 363\)](#).

### Topics

- [Named Entity Recognition \(p. 188\)](#)
- [Text Classification \(Single Label\) \(p. 191\)](#)
- [Text Classification \(Multi-label\) \(p. 194\)](#)

## Named Entity Recognition

To extract information from unstructured text and classify it into predefined categories, use an Amazon SageMaker Ground Truth named entity recognition (NER) labeling task. Traditionally, NER involves sifting through text data to locate noun phrases, called *named entities*, and categorizing each with a label, such as "person," "organization," or "brand." You can broaden this task to label longer spans of text and categorize those sequences with predefined labels that you specify.

When tasked with a named entity recognition labeling job, workers apply your labels to specific words or phrases within a larger text block. They choose a label, then apply it by using the cursor to highlight the part of the text to which the label applies. Workers can't apply multiple labels to the same text, and labels can't overlap.

You can create a named entity recognition labeling job using the Ground Truth section of the Amazon SageMaker console or the [CreateLabelingJob](#) operation.

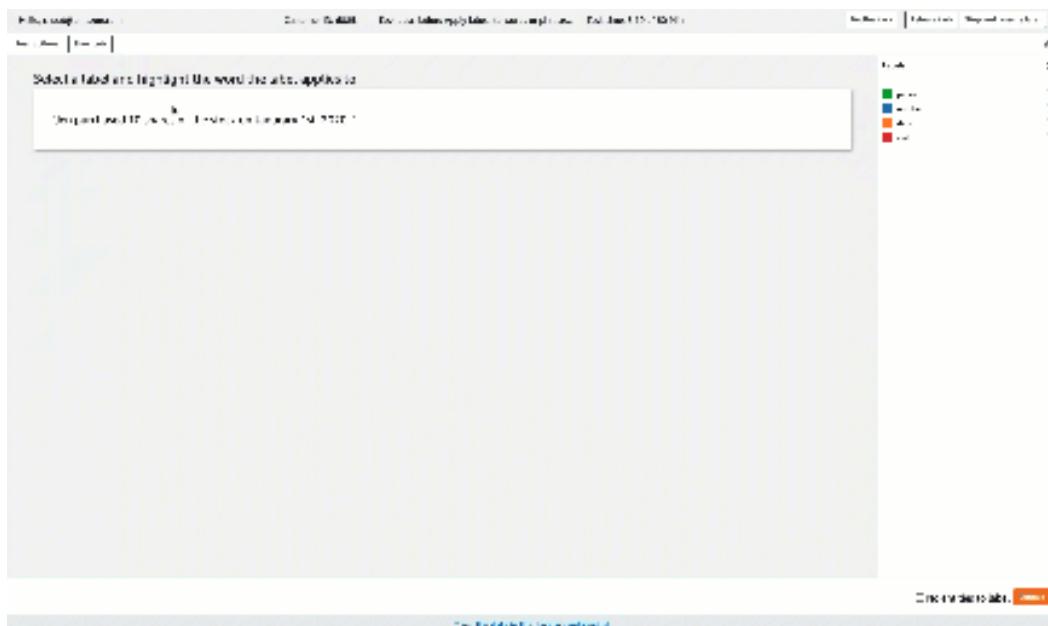
**Important**

If you manually create an input manifest file, use "source" to identify the text that you want labeled. For more information, see [Input Data \(p. 363\)](#).

## Create a Named Entity Recognition Labeling Job (Console)

You can follow the instructions [Create a Labeling Job \(Console\) \(p. 336\)](#) to learn how to create a named entity recognition labeling job in the SageMaker console. In Step 10, choose **Text** from the **Task category** drop down menu, and choose **Named entity recognition** as the task type.

Ground Truth provides a worker UI similar to the following for labeling tasks. When you create the labeling job with the console, you specify instructions to help workers complete the job and labels that workers can choose from.



## Create a Named Entity Recognition Labeling Job (API)

To create a named entity recognition labeling job, using the SageMaker API operation [CreateLabelingJob](#). This API defines this operation for all Amazon SDKs. To see a list of language-specific SDKs supported for this operation, review the **See Also** section of [CreateLabelingJob](#).

Follow the instructions on [Create a Labeling Job \(API\) \(p. 338\)](#) and do the following while you configure your request:

- Pre-annotation Lambda functions for this task type end with `PRE-NamedEntityRecognition`. To find the pre-annotation Lambda ARN for your Region, see [PreHumanTaskLambdaArn](#).
- Annotation-consolidation Lambda functions for this task type end with `ACS-NamedEntityRecognition`. To find the annotation-consolidation Lambda ARN for your Region, see [AnnotationConsolidationLambdaArn](#).

The following is an example of an [Amazon Python SDK \(Boto3\) request](#) to create a labeling job in the US East (N. Virginia) Region. All parameters in red should be replaced with your specifications and resources.

```
response = client.create_labeling_job(  
    LabelingJobName='example-ner-labeling-job',
```

```

LabelAttributeName='label',
InputConfig={
    'DataSource': {
        'S3DataSource': {
            'ManifestS3Uri': 's3://bucket/path/manifest-with-input-data.json'
        }
    },
    'DataAttributes': {
        'ContentClassifiers': [
            'FreeOfPersonallyIdentifiableInformation' | 'FreeofAdultContent',
        ]
    }
},
OutputConfig={
    'S3OutputPath': 's3://bucket/path/file-to-store-output-data',
    'KmsKeyId': 'string'
},
RoleArn='arn:aws:iam::*:role/*',
LabelCategoryConfigS3Uri='s3://bucket/path/label-categories.json',
StoppingConditions={
    'MaxHumanLabeledObjectCount': 123,
    'MaxPercentageOfInputDatasetLabeled': 123
},
HumanTaskConfig={
    'WorkteamArn': 'arn:aws:sagemaker:region*:workteam/private-crowd/*',
    'UiConfig': {
        'UiTemplateS3Uri': 's3://bucket/path/worker-task-template.html'
    },
    'PreHumanTaskLambdaArn': 'arn:aws:lambda:us-east-1:432418664414:function:PRE-
NamedEntityRecognition,
    'TaskKeywords': [
        'Named entity Recognition',
    ],
    'TaskTitle': 'Named entity Recognition task',
    'TaskDescription': 'Apply the labels provided to specific words or phrases within
the larger text block.',
    'NumberOfHumanWorkersPerDataObject': 123,
    'TaskTimeLimitInSeconds': 123,
    'TaskAvailabilityLifetimeInSeconds': 123,
    'MaxConcurrentTaskCount': 123,
    'AnnotationConsolidationConfig': {
        'AnnotationConsolidationLambdaArn': 'arn:aws:lambda:us-
east-1:432418664414:function:ACS-NamedEntityRecognition'
    },
    Tags=[{
        {
            'Key': 'string',
            'Value': 'string'
        },
    ],
}
)

```

### Provide a Template for Named Entity Recognition Labeling Jobs

If you create a labeling job using the API, you must supply a worker task template in `UiTemplateS3Uri`. Copy and modify the following template. Only modify the `short-instructions`, `full-instructions`, and `header`.

Upload this template to S3, and provide the S3 URI for this file in `UiTemplateS3Uri`.

```

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-entity-annotation
    name="crowd-entity-annotation"
    header="please classify"

```

```
    labels="{{ task.input.labels | to_json | escape }}"
    text="{{ task.input.taskObject }}"
>
<full-instructions header="Named entity recognition instructions">
  <ol><li><strong>Read</strong> the text carefully.</li>
  <li><strong>Highlight</strong> words, phrases, or sections of the text.</li>
  <li><strong>Choose</strong> the label that best matches what you have highlighted.</li>
  <li>To <strong>change</strong> a label, choose highlighted text and select a new
label.</li>
  <li>To <strong>remove</strong> a label from highlighted text, choose the X next to the
abbreviated label name on the highlighted text.</li>
  <li>You can select all of a previously highlighted text, but not a portion of it.</li>
</ol>
</full-instructions>
<short-instructions>
  <p>Enter description of the labels that workers have to choose from</p>
  <p><br></p><p><br></p>
  <p>Add examples to help workers understand the label</p>
  <p><br></p><p><br></p><p><br></p><p><br></p><p><br></p><p><br></p>
</short-instructions>
</crowd-entity-annotation>
```

## Named Entity Recognition Output Data

Once you have created a named entity recognition labeling job, your output data will be located in the Amazon S3 bucket specified in the `S3OutputPath` parameter when using the API or in the **Output dataset location** field of the **Job overview** section of the console.

To learn more about the output manifest file generated by Ground Truth and the file structure the Ground Truth uses to store your output data, see [Output Data \(p. 404\)](#).

## Text Classification (Single Label)

To categorize articles and text into predefined categories, use text classification. For example, you can use text classification to identify the sentiment conveyed in a review or the emotion underlying a section of text. Use Amazon SageMaker Ground Truth text classification to have workers sort text into categories that you define.

You create a text classification labeling job using the Ground Truth section of the Amazon SageMaker console or the [CreateLabelingJob](#) operation.

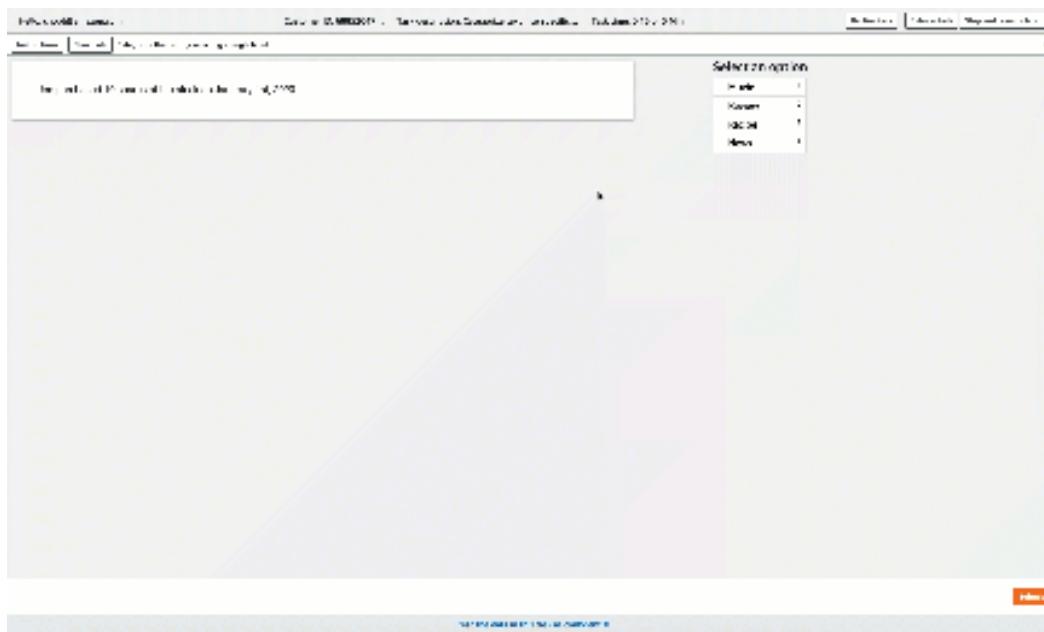
### Important

If you manually create an input manifest file, use "source" to identify the text that you want labeled. For more information, see [Input Data \(p. 363\)](#).

## Create a Text Classification Labeling Job (Console)

You can follow the instructions [Create a Labeling Job \(Console\) \(p. 336\)](#) to learn how to create a text classification labeling job in the SageMaker console. In Step 10, choose **Text** from the **Task category** drop down menu, and choose **Text Classification (Single Label)** as the task type.

Ground Truth provides a worker UI similar to the following for labeling tasks. When you create the labeling job with the console, you specify instructions to help workers complete the job and labels that workers can choose from.



## Create a Text Classification Labeling Job (API)

To create a text classification labeling job, use the SageMaker API operation `CreateLabelingJob`. This API defines this operation for all Amazon SDKs. To see a list of language-specific SDKs supported for this operation, review the **See Also** section of [CreateLabelingJob](#).

Follow the instructions on [Create a Labeling Job \(API\) \(p. 338\)](#) and do the following while you configure your request:

- Pre-annotation Lambda functions for this task type end with `PRE-TextMultiClass`. To find the pre-annotation Lambda ARN for your Region, see [PreHumanTaskLambdaArn](#).
- Annotation-consolidation Lambda functions for this task type end with `ACS-TextMultiClass`. To find the annotation-consolidation Lambda ARN for your Region, see [AnnotationConsolidationLambdaArn](#).

The following is an example of an [Amazon Python SDK \(Boto3\) request](#) to create a labeling job in the US East (N. Virginia) Region. All parameters in red should be replaced with your specifications and resources.

```
response = client.create_labeling_job(
    LabelingJobName='example-text-classification-labeling-job',
    LabelAttributeName='label',
    InputConfig={
        'DataSource': {
            'S3DataSource': {
                'ManifestS3Uri': 's3://bucket/path/manifest-with-input-data.json'
            }
        },
        'DataAttributes': {
            'ContentClassifiers': [
                'FreeOfPersonallyIdentifiableInformation' | 'FreeOfAdultContent',
            ]
        }
    },
    OutputConfig={
        'S3OutputPath': 's3://bucket/path/file-to-store-output-data',
        'KmsKeyId': 'string'
    },
}
```

```

RoleArn='arn:aws:iam::*:role/*',
LabelCategoryConfigS3Uri='s3://bucket/path/label-categories.json',
StoppingConditions={
    'MaxHumanLabeledObjectCount': 123,
    'MaxPercentageOfInputDatasetLabeled': 123
},
HumanTaskConfig={
    'WorkteamArn': 'arn:aws:sagemaker:region*:workteam/private-crowd/*',
    'UiConfig': {
        'UiTemplateS3Uri': 's3://bucket/path/worker-task-template.html'
    },
    'PreHumanTaskLambdaArn': 'arn:aws:lambda:us-east-1:432418664414:function:PRE-
TextMultiClass,
    'TaskKeywords': [
        'Text classification',
    ],
    'TaskTitle': 'Text classification task',
    'TaskDescription': 'Carefully read and classify this text using the categories
provided',
    'NumberOfHumanWorkersPerDataObject': 123,
    'TaskTimeLimitInSeconds': 123,
    'TaskAvailabilityLifetimeInSeconds': 123,
    'MaxConcurrentTaskCount': 123,
    'AnnotationConsolidationConfig': {
        'AnnotationConsolidationLambdaArn': 'arn:aws:lambda:us-
east-1:432418664414:function:ACS-TextMultiClass'
    },
    Tags:[
        {
            'Key': 'string',
            'Value': 'string'
        },
    ],
}
)

```

### Provide a Template for Text Classification Labeling Jobs

If you create a labeling job using the API, you must supply a worker task template in `UiTemplateS3Uri`. Copy and modify the following template. Only modify the `short-instructions`, `full-instructions`, and header.

Upload this template to S3, and provide the S3 URI for this file in `UiTemplateS3Uri`.

```

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
<crowd-classifier
    name="crowd-classifier"
    categories="{{ task.input.labels | to_json | escape }}"
    header="classify text"
>
    <classification-target style="white-space: pre-wrap">
        {{ task.input.taskObject }}
    </classification-target>
    <full-instructions header="Classifier instructions">
        <ol><li><strong>Read</strong> the text carefully.</li>
        <li><strong>Read</strong> the examples to understand more about the options.</li>
        <li><strong>Choose</strong> the appropriate labels that best suit the text.</li></ol>
    </full-instructions>
    <short-instructions>
        <p>Enter description of the labels that workers have to choose from</p>
        <p><br></p><p><br></p><p>Add examples to help workers understand the label</p>
        <p><br></p><p><br></p><p><br></p><p><br></p><p><br></p><p><br></p><p><br></p>
    </short-instructions>
</crowd-classifier>

```

```
</crowd-form>
```

## Text Classification Output Data

Once you have created a text classification labeling job, your output data will be located in the Amazon S3 bucket specified in the `S3OutputPath` parameter when using the API or in the **Output dataset location** field of the **Job overview** section of the console.

To learn more about the output manifest file generated by Ground Truth and the file structure the Ground Truth uses to store your output data, see [Output Data \(p. 404\)](#).

To see an example of an output manifest files from a text classification labeling job, see [Classification Job Output \(p. 408\)](#).

## Text Classification (Multi-label)

To categorize articles and text into multiple predefined categories, use the multi-label text classification task type. For example, you can use this task type to identify more than one emotion conveyed in text.

When working on a multi-label text classification task, workers should choose all applicable labels, but must choose at least one. When creating a job using this task type, you can provide up to 50 label categories.

Amazon SageMaker Ground Truth doesn't provide a "none" category for when none of the labels applies. To provide this option to workers, include a label similar to "none" or "other" when you create a multi-label text classification job.

To restrict workers to choosing a single label for each document or text selection, use the [Text Classification \(Single Label\) \(p. 191\)](#) task type.

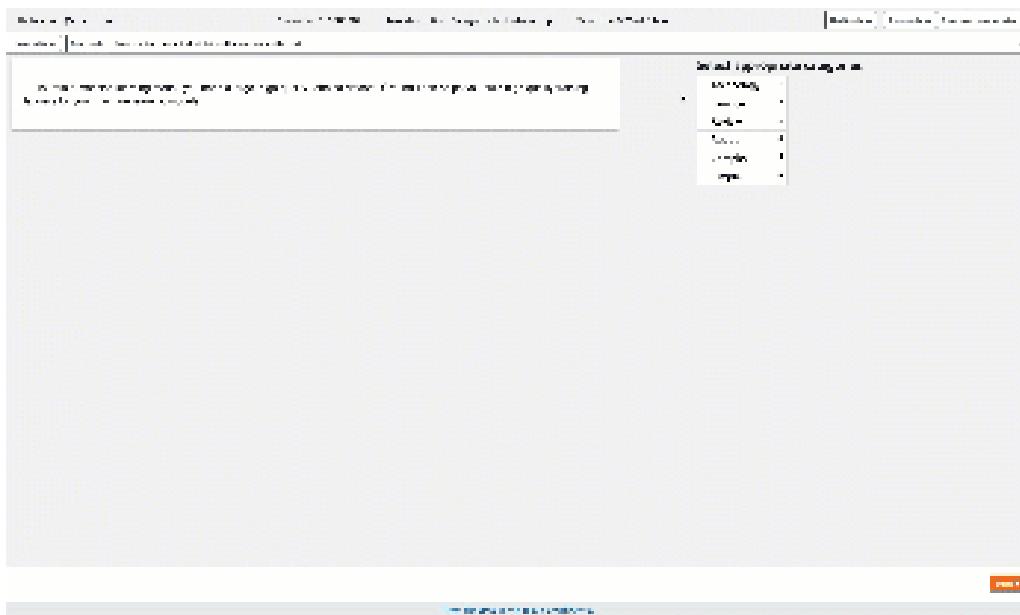
**Important**

If you manually create an input manifest file, use "source" to identify the text that you want labeled. For more information, see [Input Data \(p. 363\)](#).

## Create a Multi-Label Text Classification Labeling Job (Console)

You can follow the instructions [Create a Labeling Job \(Console\) \(p. 336\)](#) to learn how to create a multi-label text classification labeling job in the Amazon SageMaker console. In Step 10, choose **Text** from the **Task category** drop down menu, and choose **Text Classification (Multi-label)** as the task type.

Ground Truth provides a worker UI similar to the following for labeling tasks. When you create the labeling job with the console, you specify instructions to help workers complete the job and labels that workers can choose from.



## Create a Multi-Label Text Classification Labeling Job (API)

To create a multi-label text classification labeling job, use the SageMaker API operation `CreateLabelingJob`. This API defines this operation for all Amazon SDKs. To see a list of language-specific SDKs supported for this operation, review the **See Also** section of [CreateLabelingJob](#).

Follow the instructions on [Create a Labeling Job \(API\) \(p. 338\)](#) and do the following while you configure your request:

- Pre-annotation Lambda functions for this task type end with `PRE-TextMultiClassMultiLabel`. To find the pre-annotation Lambda ARN for your Region, see [PreHumanTaskLambdaArn](#).
- Annotation-consolidation Lambda functions for this task type end with `ACS-TextMultiClassMultiLabel`. To find the annotation-consolidation Lambda ARN for your Region, see [AnnotationConsolidationLambdaArn](#).

The following is an example of an [Amazon Python SDK \(Boto3\) request](#) to create a labeling job in the US East (N. Virginia) Region. All parameters in red should be replaced with your specifications and resources.

```
response = client.create_labeling_job(
    LabelingJobName='example-multi-label-text-classification-labeling-job',
    LabelAttributeName='label',
    InputConfig={
        'DataSource': {
            'S3DataSource': {
                'ManifestS3Uri': 's3://bucket/path/manifest-with-input-data.json'
            }
        },
        'DataAttributes': {
            'ContentClassifiers': [
                'FreeOfPersonallyIdentifiableInformation' | 'FreeOfAdultContent',
            ]
        }
    },
    OutputConfig={
        'S3OutputPath': 's3://bucket/path/file-to-store-output-data',
        'KmsKeyId': 'string'
    },
}
```

```

RoleArn='arn:aws:iam::*:role/*',
LabelCategoryConfigS3Uri='s3://bucket/path/label-categories.json',
StoppingConditions={
    'MaxHumanLabeledObjectCount': 123,
    'MaxPercentageOfInputDatasetLabeled': 123
},
HumanTaskConfig={
    'WorkteamArn': 'arn:aws:sagemaker:region*:workteam/private-crowd/*',
    'UiConfig': {
        'UiTemplateS3Uri': 's3://bucket/path/custom-worker-task-template.html'
    },
    'PreHumanTaskLambdaArn': 'arn:aws:lambda:function:PRE-TextMultiClassMultiLabel',
    'TaskKeywords': [
        'Text Classification',
    ],
    'TaskTitle': 'Multi-label text classification task',
    'TaskDescription': 'Select all labels that apply to the text shown',
    'NumberOfHumanWorkersPerDataObject': 123,
    'TaskTimeLimitInSeconds': 123,
    'TaskAvailabilityLifetimeInSeconds': 123,
    'MaxConcurrentTaskCount': 123,
    'AnnotationConsolidationConfig': {
        'AnnotationConsolidationLambdaArn': 'arn:aws:lambda:us-
east-1:432418664414:function:ACS-TextMultiClassMultiLabel'
    },
    Tags:[
    {
        'Key': 'string',
        'Value': 'string'
    },
    ]
}
)

```

### Create a Template for Multi-label Text Classification

If you create a labeling job using the API, you must supply a worker task template in `UiTemplateS3Uri`. Copy and modify the following template. Only modify the `short-instructions`, `full-instructions`, and `header`.

Upload this template to S3, and provide the S3 URI for this file in `UiTemplateS3Uri`.

```

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
    <crowd-classifier-multi-select
        name="crowd-classifier-multi-select"
        categories="{{ task.input.labels | to_json | escape }}"
        header="Please identify all classes in the below text"
    >
        <classification-target style="white-space: pre-wrap">
            {{ task.input.taskObject }}
        </classification-target>
        <full-instructions header="Classifier instructions">
            <ol><li><strong>Read</strong> the text carefully.</li>
            <li><strong>Read</strong> the examples to understand more about the options.</li>
            <li><strong>Choose</strong> the appropriate labels that best suit the text.</li>
        </ol>
    </full-instructions>
    <short-instructions>
        <p>Enter description of the labels that workers have to choose from</p>
        <p><br></p>
        <p><br></p><p>Add examples to help workers understand the label</p>
        <p><br></p><p><br></p><p><br></p><p><br></p><p><br></p><p><br></p><p><br></p>
    </short-instructions>
</crowd-classifier-multi-select>
</crowd-form>

```

To learn how to create a custom template, see [Creating Custom Labeling Workflows \(p. 300\)](#).

## Multi-label Text Classification Output Data

Once you have created a multi-label text classification labeling job, your output data will be located in the Amazon S3 bucket specified in the `S3OutputPath` parameter when using the API or in the **Output dataset location** field of the **Job overview** section of the console.

To learn more about the output manifest file generated by Ground Truth and the file structure the Ground Truth uses to store your output data, see [Output Data \(p. 404\)](#).

To see an example of output manifest files for multi-label text classification labeling job, see [Multi-label Classification Job Output \(p. 408\)](#).

# Label Videos and Video Frames

You can use Ground Truth to classify videos and annotate video frames (still images extracted from videos) using one of the three built-in video task types. These task types streamline the process of creating video and video frame labeling jobs using the Amazon SageMaker console, API, and language-specific SDKs.

- **Video clip classification** – Enable workers to classify videos into categories you specify. For example, you can use this task type to have workers categorize videos into topics like sports, comedy, music, and education. To learn more, see [Video Classification \(p. 197\)](#).
- **Video frame labeling jobs** – Enable workers to annotate video frames extracted from a video using bounding boxes, polylines, polygons or keypoint annotation tools. Ground Truth offers two built-in task types to label video frames:
  - *Video frame object detection*: Enable workers to identify and locate objects in video frames.
  - *Video frame object tracking*: Enable workers to track the movement of objects across video frames.
  - *Video frame adjustment jobs*: Have workers adjust labels, label category attributes, and frame attributes from a previous video frame object detection or object tracking labeling job.
  - *Video frame verification jobs*: Have workers verify labels, label category attributes, and frame attributes from a previous video frame object detection or object tracking labeling job.

If you have video files, you can use the Ground Truth automatic frame extraction tool to extract video frames from your videos. To learn more, see [Video Frame Input Data \(p. 398\)](#).

### Tip

To learn more about supported file types and input data quotas, see [Input Data \(p. 363\)](#).

### Topics

- [Video Classification \(p. 197\)](#)
- [Label Video Frames \(p. 202\)](#)
- [Worker Instructions \(p. 214\)](#)

# Video Classification

Use an Amazon SageMaker Ground Truth video classification labeling task when you need workers to classify videos using predefined labels that you specify. Workers are shown videos and are asked to choose one label for each video.

You create a video classification labeling job using the Ground Truth section of the Amazon SageMaker console or the `CreateLabelingJob` operation.

Your video files must be encoded in a format that is supported by the browser used by the work team that labels your data. It is recommended that you verify that all video file formats in your input manifest file display correctly using the worker UI preview. You can communicate supported browsers to your workers using worker instructions. To see supported file formats, see [Supported Data Formats \(p. 366\)](#).

**Important**

For this task type, if you create your own manifest file, use "source-ref" to identify the location of each video file in Amazon S3 that you want labeled. For more information, see [Input Data \(p. 363\)](#).

## Create a Video Classification Labeling Job (Console)

You can follow the instructions in [Create a Labeling Job \(Console\) \(p. 336\)](#) to learn how to create a video classification labeling job in the SageMaker console. In step 10, choose **Video** from the **Task category** dropdown list, and choose **Video Classification** as the task type.

Ground Truth provides a worker UI similar to the following for labeling tasks. When you create a labeling job in the console, you specify instructions to help workers complete the job and labels from which workers can choose.

## Instructions

X

[View full instructions](#)

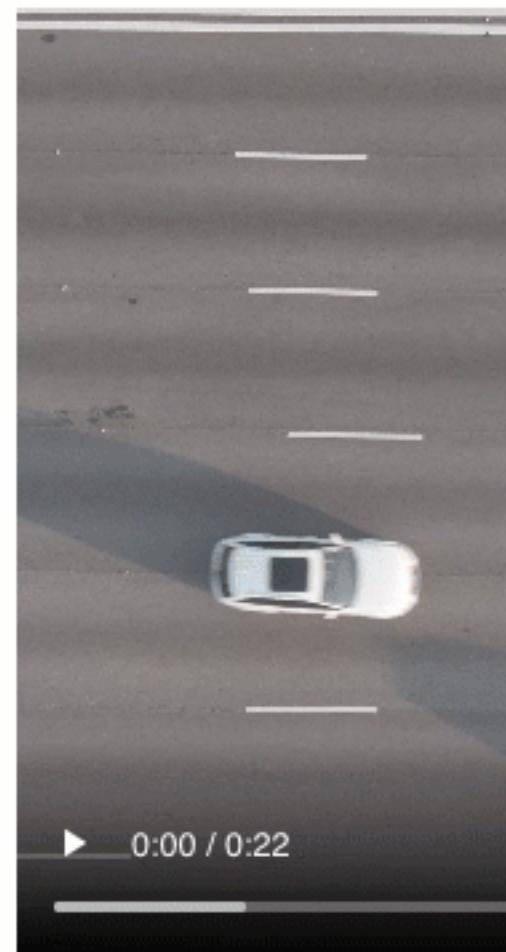
[View tool guide](#)

**Select a single label that best describes this video clip.**

**Select none of the above if none of the other labels apply.**

**Select Submit when you are done.**

## Watch and then class...



## Create a Video Classification Labeling Job (API)

This section covers details you need to know when you create a labeling job using the SageMaker API operation `CreateLabelingJob`. This API defines this operation for all Amazon SDKs. To see a list of language-specific SDKs supported for this operation, review the **See Also** section of [CreateLabelingJob](#).

Follow the instructions on [Create a Labeling Job \(API\) \(p. 338\)](#) and do the following while you configure your request:

- Use a pre-annotation Lambda function that ends with `PRE-VideoClassification`. To find the pre-annotation Lambda ARN for your Region, see [PreHumanTaskLambdaArn](#).
- Use an annotation-consolidation Lambda function that ends with `ACS-VideoClassification`. To find the annotation-consolidation Lambda ARN for your Region, see [AnnotationConsolidationLambdaArn](#).

The following is an example of an [Amazon Python SDK \(Boto3\) request](#) to create a labeling job in the US East (N. Virginia) Region.

```
response = client.create_labeling_job(
    LabelingJobName='example-video-classification-labeling-job',
    LabelAttributeName='label',
    InputConfig={
        'DataSource': {
            'S3DataSource': {
                'ManifestS3Uri': 's3://bucket/path/manifest-with-input-data.json'
            }
        },
        'DataAttributes': {
            'ContentClassifiers': [
                'FreeOfPersonallyIdentifiableInformation' | 'FreeOfAdultContent',
            ]
        }
    },
    OutputConfig={
        'S3OutputPath': 's3://bucket/path/file-to-store-output-data',
        'KmsKeyId': 'string'
    },
    RoleArn='arn:aws:iam::*:role/*',
    LabelCategoryConfigS3Uri='s3://bucket/path/label-categories.json',
    StoppingConditions={
        'MaxHumanLabeledObjectCount': 123,
        'MaxPercentageOfInputDatasetLabeled': 123
    },
    HumanTaskConfig={
        'WorkteamArn': 'arn:aws:sagemaker:region::workteam/private-crowd/*',
        'UiConfig': {
            'UiTemplateS3Uri': 's3://bucket/path/worker-task-template.html'
        },
        'PreHumanTaskLambdaArn': 'arn:aws:lambda:us-east-1:432418664414:function:PRE-VideoClassification',
        'TaskKeywords': [
            'Video Classification',
        ],
        'TaskTitle': 'Video classification task',
        'TaskDescription': 'Select a label to classify this video',
        'NumberOfHumanWorkersPerDataObject': 123,
        'TaskTimeLimitInSeconds': 123,
        'TaskAvailabilityLifetimeInSeconds': 123,
        'MaxConcurrentTaskCount': 123,
        'AnnotationConsolidationConfig': {
    }
```

```

        'AnnotationConsolidationLambdaArn': 'arn:aws:lambda:us-
east-1:432418664414:function:ACS-VideoClassification'
    },
    Tags:[
    {
        'Key': 'string',
        'Value': 'string'
    },
]
)

```

## Provide a Template for Video Classification

If you create a labeling job using the API, you must supply a worker task template in `UiTemplateS3Uri`. Copy and modify the following template by modifying the `short-instructions`, `full-instructions`, and `header`. Upload this template to Amazon S3, and provide the Amazon S3 URI to this file in `UiTemplateS3Uri`.

```

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
    <crowd-classifier
        name="crowd-classifier"
        categories="{{ task.input.labels | to_json | escape }}"
        header="Please classify video"
    >
        <classification-target>
            <video width="100%" controls/>
                <source src="{{ task.input.taskObject | grant_read_access }}"
type="video/mp4"/>
                <source src="{{ task.input.taskObject | grant_read_access }}"
type="video/webm"/>
                <source src="{{ task.input.taskObject | grant_read_access }}"
type="video/ogg"/>
            Your browser does not support the video tag.
        </video>
    </classification-target>
    <full-instructions header="Video classification instructions">
        <ol><li><strong>Read</strong> the task carefully and inspect the
video.</li>
        <li><strong>Read</strong> the options and review the examples
provided to understand more about the labels.</li>
        <li><strong>Choose</strong> the appropriate label that best suits
the video.</li></ol>
    </full-instructions>
    <short-instructions>
        <h3><span style="color: #0000ff;">Good example</span></h3>
        <p>Enter description to explain the correct label to the workers</p>
        <p></p>
        <h3><span style="color: #cc0000;">Bad example</span></h3>
        <p>Enter description of an incorrect label</p>
        <p></p>
    </short-instructions>
    </crowd-classifier>
</crowd-form>

```

## Video Classification Output Data

Once you have created a video classification labeling job, your output data is located in the Amazon S3 bucket specified in the `S3OutputPath` parameter when using the API or in the **Output dataset location** field of the **Job overview** section of the console.

To learn more about the output manifest file generated by Ground Truth and the file structure the Ground Truth uses to store your output data, see [Output Data \(p. 404\)](#).

To see an example of output manifest files for video classification labeling jobs, see [Classification Job Output \(p. 408\)](#).

## Label Video Frames

You can use Ground Truth built-in video frame task types to have workers annotate video frames using bounding boxes, polylines, polygons or keypoints. A *video frame* is a sequence of images that have been extracted from a video.

If you do not have video frames, you can provide video files (MP4 files) and use the Ground Truth automated frame extraction tool to extract video frames. To learn more, see [Provide Video Files \(p. 400\)](#).

You can use the following built-in video task types to create video frame labeling jobs using the Amazon SageMaker console, API, and language-specific SDKs.

- **Video frame object detection** – Use this task type when you want workers to identify and locate objects in sequences of video frames. You provide a list of categories, and workers can select one category at a time and annotate objects which the category applies to in all frames. For example, you can use this task to ask workers to identify and localize various objects in a scene, such as cars, bikes, and pedestrians.
- **Video frame object tracking** – Use this task type when you want workers to track the movement of instances of objects across sequences of video frames. When a worker adds an annotation to a single frame, that annotation is associated with a unique instance ID. The worker adds annotations associated with the same ID in all other frames to identify the same object or person. For example, a worker can track the movement of a vehicle across a sequences of video frames by drawing bounding boxes associated with the same ID around the vehicle in each frame that it appears.

Use the following topics to learn more about these built-in task types and to how to create a labeling job using each task type. See [Task Types \(p. 211\)](#) to learn more about the annotations tools (bounding boxes, polylines, polygons and keypoints) available for these task types.

Before you create a labeling job, we recommend that you review [Video Frame Labeling Job Overview \(p. 210\)](#).

### Topics

- [Video Frame Object Detection \(p. 202\)](#)
- [Video Frame Object Tracking \(p. 206\)](#)
- [Video Frame Labeling Job Overview \(p. 210\)](#)

## Video Frame Object Detection

You can use the video frame object detection task type to have workers identify and locate objects in a sequence of video frames (images extracted from a video) using bounding boxes, polylines, polygons

or keypoint *annotation tools*. The tool you choose defines the video frame task type you create. For example, you can use a bounding box video frame object detection task type workers to identify and localize various objects in a series of video frames, such as cars, bikes, and pedestrians.

You can create a video frame object detection labeling job using the Amazon SageMaker Ground Truth console, the SageMaker API, and language-specific Amazon SDKs. To learn more, see [Create a Video Frame Object Detection Labeling Job \(p. 203\)](#) and select your preferred method. See [Task Types \(p. 211\)](#) to learn more about the annotations tools you can choose from when you create a labeling job.

Ground Truth provides a worker UI and tools to complete your labeling job tasks: [Preview the Worker UI \(p. 203\)](#).

You can create a job to adjust annotations created in a video object detection labeling job using the video object detection adjustment task type. To learn more, see [Create Video Frame Object Detection Adjustment or Verification Labeling Job \(p. 206\)](#).

### [Preview the Worker UI](#)

Ground Truth provides workers with a web user interface (UI) to complete your video frame object detection annotation tasks. You can preview and interact with the worker UI when you create a labeling job in the console. If you are a new user, we recommend that you create a labeling job through the console using a small input dataset to preview the worker UI and ensure your video frames, labels, and label attributes appear as expected.

The UI provides workers with the following assistive labeling tools to complete your object detection tasks:

- For all tasks, workers can use the **Copy to next** and **Copy to all** features to copy an annotation to the next frame or to all subsequent frames respectively.
- For tasks that include the bounding box tools, workers can use a **Predict next** feature to draw a bounding box in a single frame, and then have Ground Truth predict the location of boxes with the same label in all other frames. Workers can then make adjustments to correct predicted box locations.

### [Create a Video Frame Object Detection Labeling Job](#)

You can create a video frame object detection labeling job using the SageMaker console or the [CreateLabelingJob](#) API operation.

This section assumes that you have reviewed the [Video Frame Labeling Job Overview \(p. 210\)](#) and have chosen the type of input data and the input dataset connection you are using.

### [Create a Labeling Job \(Console\)](#)

You can follow the instructions in [Create a Labeling Job \(Console\) \(p. 336\)](#) to learn how to create a video frame object tracking job in the SageMaker console. In step 10, choose **Video - Object detection** from the **Task category** dropdown list. Select the task type you want by selecting one of the cards in **Task selection**.

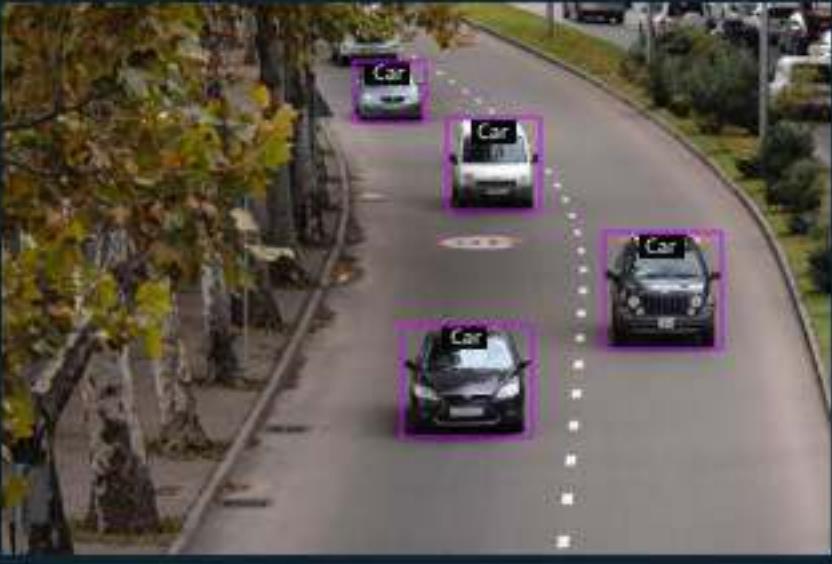
## Task type [Info](#)

**Task category**  
Select the type of data being labeled to view available task templates for it or select 'Create new' to create a custom task template.

**Video - Object detection**

**Task selection**  
Select the task that a human worker will perform to label objects in your dataset.

**Bounding box**  
Get workers to draw bounding boxes around specified objects in your video. [Info](#)



**Polyline**  
Get workers to draw polyline around specified objects in your video. [Info](#)

**Polyline**  
Get workers to draw polyline around specified objects in your video. [Info](#)



**Polyline**  
Get workers to draw polyline around specified objects in your video. [Info](#)

**Keyframe**  
Get workers to draw keyframe around specified objects in your video. [Info](#)

## Create a Labeling Job (API)

You create an object detection labeling job using the SageMaker API operation `CreateLabelingJob`. This API defines this operation for all Amazon SDKs. To see a list of language-specific SDKs supported for this operation, review the **See Also** section of [CreateLabelingJob](#).

[Create a Labeling Job \(API\) \(p. 338\)](#) provides an overview of the `CreateLabelingJob` operation. Follow these instructions and do the following while you configure your request:

- You must enter an ARN for `HumanTaskUiArn`. Use `arn:aws:sagemaker:<region>:394669845002:human-task-ui/VideoObjectDetection`. Replace `<region>` with the Amazon Region in which you are creating the labeling job.  
  
Do not include an entry for the `UiTemplateS3Uri` parameter.
- Your `LabelAttributeName` must end in `-ref`. For example, `video-od-labels-ref`.
- Your input manifest file must be a video frame sequence manifest file. You can create this manifest file using the SageMaker console, or create it manually and upload it to Amazon S3. For more information, see [Input Data Setup \(p. 400\)](#).
- You can only use private or vendor work teams to create video frame object detection labeling jobs.
- You specify your labels, label category and frame attributes, the task type, and worker instructions in a label category configuration file. Specify the task type (bounding boxes, polylines, polygons or keypoint) using `annotationType` in your label category configuration file. For more information, see [Create a Labeling Category Configuration File with Label Category and Frame Attributes \(p. 348\)](#) to learn how to create this file.
- You need to provide pre-defined ARNs for the pre-annotation and post-annotation (ACS) Lambda functions. These ARNs are specific to the Amazon Region you use to create your labeling job.
  - To find the pre-annotation Lambda ARN, refer to `PreHumanTaskLambdaArn`. Use the Region in which you are creating your labeling job to find the correct ARN that ends with `PRE-VideoObjectDetection`.
  - To find the post-annotation Lambda ARN, refer to `AnnotationConsolidationLambdaArn`. Use the Region in which you are creating your labeling job to find the correct ARN that ends with `ACS-VideoObjectDetection`.
- The number of workers specified in `NumberOfHumanWorkersPerDataObject` must be 1.
- Automated data labeling is not supported for video frame labeling jobs. Do not specify values for parameters in `LabelingJobAlgorithmsConfig`.
- Video frame object tracking labeling jobs can take multiple hours to complete. You can specify a longer time limit for these labeling jobs in `TaskTimeLimitInSeconds` (up to 7 days, or 604,800 seconds).

The following is an example of an [Amazon Python SDK \(Boto3\) request](#) to create a labeling job in the US East (N. Virginia) Region.

```
response = client.create_labeling_job(
    LabelingJobName='example-video-od-labeling-job',
    LabelAttributeName='label',
    InputConfig={
        'DataSource': {
            'S3DataSource': {
                'ManifestS3Uri': 's3://path/video-frame-sequence-input-manifest.json'
            }
        },
        'DataAttributes': {
            'ContentClassifiers': [
                'FreeOfPersonallyIdentifiableInformation' | 'FreeOfAdultContent',
            ]
        }
    }
)
```

```

},
OutputConfig={
    'S3OutputPath': 's3://prefix/file-to-store-output-data',
    'KmsKeyId': 'string'
},
RoleArn='arn:aws:iam::*:role/*',
LabelCategoryConfigS3Uri='s3://bucket/prefix/label-categories.json',
StoppingConditions={
    'MaxHumanLabeledObjectCount': 123,
    'MaxPercentageOfInputDatasetLabeled': 123
},
HumanTaskConfig={
    'WorkteamArn': 'arn:aws:sagemaker:us-east-1:*:workteam/private-crowd/*',
    'UiConfig': {
        'HumanTaskUiArn': 'arn:aws:sagemaker:us-east-1:394669845002:human-task-ui/
VideoObjectDetection'
    },
    'PreHumanTaskLambdaArn': 'arn:aws:lambda:us-east-1:432418664414:function:PRE-
VideoObjectDetection',
    'TaskKeywords': [
        'Video Frame Object Detection',
    ],
    'TaskTitle': 'Video frame object detection task',
    'TaskDescription': 'Classify and identify the location of objects and people in
video frames',
    'NumberOfHumanWorkersPerDataObject': 123,
    'TaskTimeLimitInSeconds': 123,
    'TaskAvailabilityLifetimeInSeconds': 123,
    'MaxConcurrentTaskCount': 123,
    'AnnotationConsolidationConfig': {
        'AnnotationConsolidationLambdaArn': 'arn:aws:lambda:us-
east-1:432418664414:function:ACS-VideoObjectDetection'
    },
    Tags:[
    {
        'Key': 'string',
        'Value': 'string'
    },
    ]
}
)

```

## Create Video Frame Object Detection Adjustment or Verification Labeling Job

You can create an adjustment and verification labeling job using the Ground Truth console or `CreateLabelingJob` API. To learn more about adjustment and verification labeling jobs, and to learn how create one, see [Verify and Adjust Labels \(p. 293\)](#).

### Output Data Format

When you create a video frame object detection labeling job, tasks are sent to workers. When these workers complete their tasks, labels are written to the Amazon S3 output location you specified when you created the labeling job. To learn about the video frame object detection output data format, see [Video Frame Object Detection Output \(p. 416\)](#). If you are a new user of Ground Truth, see [Output Data \(p. 404\)](#) to learn more about the Ground Truth output data format.

## Video Frame Object Tracking

You can use the video frame object tracking task type to have workers track the movement of objects in a sequence of video frames (images extracted from a video) using bounding boxes, polylines, polygons or keypoint annotation tools. The tool you choose defines the video frame task type you create. For example, you can use a bounding box video frame object tracking task type to ask workers to track the movement of objects, such as cars, bikes, and pedestrians by drawing boxes around them.

You provide a list of categories, and each annotation that a worker adds to a video frame is identified as an *instance* of that category using an instance ID. For example, if you provide the label category car, the first car that a worker annotates will have the instance ID car:1. The second car the worker annotates will have the instance ID car:2. To track an object's movement, the worker adds annotations associated with the same instance ID around to object in all frames.

You can create a video frame object tracking labeling job using the Amazon SageMaker Ground Truth console, the SageMaker API, and language-specific Amazon SDKs. To learn more, see [Create a Video Frame Object Detection Labeling Job \(p. 203\)](#) and select your preferred method. See [Task Types \(p. 211\)](#) to learn more about the annotations tools you can choose from when you create a labeling job.

Ground Truth provides a worker UI and tools to complete your labeling job tasks: [Preview the Worker UI \(p. 203\)](#).

You can create a job to adjust annotations created in a video object detection labeling job using the video object detection adjustment task type. To learn more, see [Create Video Frame Object Detection Adjustment or Verification Labeling Job \(p. 206\)](#).

### Preview the Worker UI

Ground Truth provides workers with a web user interface (UI) to complete your video frame object tracking annotation tasks. You can preview and interact with the worker UI when you create a labeling job in the console. If you are a new user, we recommend that you create a labeling job through the console using a small input dataset to preview the worker UI and ensure your video frames, labels, and label attributes appear as expected.

The UI provides workers with the following assistive labeling tools to complete your object tracking tasks:

- For all tasks, workers can use the **Copy to next** and **Copy to all** features to copy an annotation with the same unique ID to the next frame or to all subsequent frames respectively.
- For tasks that include the bounding box tools, workers can use a **Predict next** feature to draw a bounding box in a single frame, and then have Ground Truth predict the location of boxes with the same unique ID in all other frames. Workers can then make adjustments to correct predicted box locations.

### Create a Video Frame Object Tracking Labeling Job

You can create a video frame object tracking labeling job using the SageMaker console or the [CreateLabelingJob](#) API operation.

This section assumes that you have reviewed the [Video Frame Labeling Job Overview \(p. 210\)](#) and have chosen the type of input data and the input dataset connection you are using.

#### Create a Labeling Job (Console)

You can follow the instructions in [Create a Labeling Job \(Console\) \(p. 336\)](#) to learn how to create a video frame object tracking job in the SageMaker console. In step 10, choose **Video - Object tracking** from the **Task category** dropdown list. Select the task type you want by selecting one of the cards in **Task selection**.

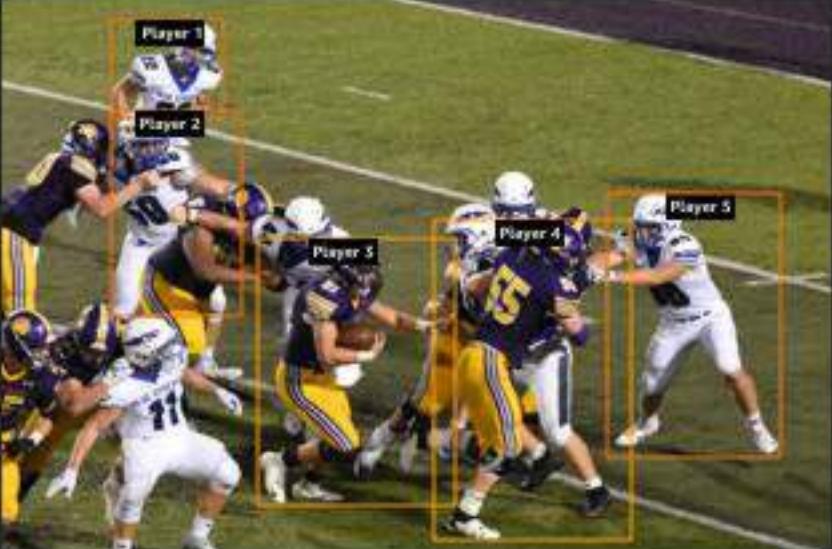
**Task type** [Info](#)

**Task category**  
Select the type of data being labeled to view available task templates for it or select 'Create new' to build a custom template.

**Video - Object tracking**

**Task selection**  
Select the task that a human worker will perform to label objects in your dataset.

**Bounding box**  
Get workers to track specific instances of objects in your video across multiple frames in your bounding boxes. [Info](#)



**Polyline**  
Get workers to track specific instances of objects in your video across multiple frames in your polylines. [Info](#)

**Polyline**  
Get workers to track specific instances of objects in your video across multiple frames in your polylines. [Info](#)



## Create a Labeling Job (API)

You create an object tracking labeling job using the SageMaker API operation `CreateLabelingJob`. This API defines this operation for all Amazon SDKs. To see a list of language-specific SDKs supported for this operation, review the **See Also** section of [CreateLabelingJob](#).

[Create a Labeling Job \(API\) \(p. 338\)](#) provides an overview of the `CreateLabelingJob` operation. Follow these instructions and do the following while you configure your request:

- You must enter an ARN for `HumanTaskUiArn`. Use `arn:aws:sagemaker:<region>:394669845002:human-task-ui/VideoObjectTracking`. Replace `<region>` with the Amazon Region in which you are creating the labeling job.

Do not include an entry for the `UiTemplateS3Uri` parameter.
- Your `LabelAttributeName` must end in `-ref`. For example, `ot-labels-ref`.
- Your input manifest file must be a video frame sequence manifest file. You can create this manifest file using the SageMaker console, or create it manually and upload it to Amazon S3. For more information, see [Input Data Setup \(p. 400\)](#). If you create a streaming labeling job, the input manifest file is optional.
- You can only use private or vendor work teams to create video frame object detection labeling jobs.
- You specify your labels, label category and frame attributes, the task type, and worker instructions in a label category configuration file. Specify the task type (bounding boxes, polylines, polygons or keypoint) using `annotationType` in your label category configuration file. For more information, see [Create a Labeling Category Configuration File with Label Category and Frame Attributes \(p. 348\)](#) to learn how to create this file.
- You need to provide pre-defined ARNs for the pre-annotation and post-annotation (ACS) Lambda functions. These ARNs are specific to the Amazon Region you use to create your labeling job.
  - To find the pre-annotation Lambda ARN, refer to `PreHumanTaskLambdaArn`. Use the Region in which you are creating your labeling job to find the correct ARN that ends with `PRE-VideoObjectTracking`.
  - To find the post-annotation Lambda ARN, refer to `AnnotationConsolidationLambdaArn`. Use the Region in which you are creating your labeling job to find the correct ARN that ends with `ACS-VideoObjectTracking`.
- The number of workers specified in `NumberOfHumanWorkersPerDataObject` must be 1.
- Automated data labeling is not supported for video frame labeling jobs. Do not specify values for parameters in `LabelingJobAlgorithmsConfig`.
- Video frame object tracking labeling jobs can take multiple hours to complete. You can specify a longer time limit for these labeling jobs in `TaskTimeLimitInSeconds` (up to 7 days, or 604,800 seconds).

The following is an example of an [Amazon Python SDK \(Boto3\) request](#) to create a labeling job in the US East (N. Virginia) Region.

```
response = client.create_labeling_job(
    LabelingJobName='example-video-ot-labeling-job',
    LabelAttributeName='label',
    InputConfig={
        'DataSource': {
            'S3DataSource': {
                'ManifestS3Uri': 's3://path/video-frame-sequence-input-manifest.json'
            }
        },
        'DataAttributes': {
            'ContentClassifiers': [
                'FreeOfPersonallyIdentifiableInformation' | 'FreeOfAdultContent',

```

```

        ]
    },
    OutputConfig={
        'S3OutputPath': 's3://prefix/file-to-store-output-data',
        'KmsKeyId': 'string'
    },
    RoleArn='arn:aws:iam::*:role/*',
    LabelCategoryConfigS3Uri='s3://bucket/prefix/label-categories.json',
    StoppingConditions={
        'MaxHumanLabeledObjectCount': 123,
        'MaxPercentageOfInputDatasetLabeled': 123
    },
    HumanTaskConfig={
        'WorkteamArn': 'arn:aws:sagemaker:us-east-1:*:workteam/private-crowd/*',
        'UiConfig': {
            'HumanTaskUiArn': 'arn:aws:sagemaker:us-east-1:394669845002:human-task-ui/
VideoObjectTracking'
        },
        'PreHumanTaskLambdaArn': 'arn:aws:lambda:us-east-1:432418664414:function:PRE-
VideoObjectTracking',
        'TaskKeywords': [
            'Video Frame Object Tracking',
        ],
        'TaskTitle': 'Video frame object tracking task',
        'TaskDescription': 'Tracking the location of objects and people across video
frames',
        'NumberOfHumanWorkersPerDataObject': 123,
        'TaskTimeLimitInSeconds': 123,
        'TaskAvailabilityLifetimeInSeconds': 123,
        'MaxConcurrentTaskCount': 123,
        'AnnotationConsolidationConfig': {
            'AnnotationConsolidationLambdaArn': 'arn:aws:lambda:us-
east-1:432418664414:function:ACS-VideoObjectTracking'
        },
        Tags:[
            {
                'Key': 'string',
                'Value': 'string'
            },
        ],
    }
)

```

## Create a Video Frame Object Tracking Adjustment or Verification Labeling Job

You can create an adjustment and verification labeling job using the Ground Truth console or CreateLabelingJob API. To learn more about adjustment and verification labeling jobs, and to learn how to create one, see [Verify and Adjust Labels \(p. 293\)](#).

### Output Data Format

When you create a video frame object tracking labeling job, tasks are sent to workers. When these workers complete their tasks, labels are written to the Amazon S3 output location you specified when you created the labeling job. To learn about the video frame object tracking output data format, see [Video Frame Object Tracking Output \(p. 418\)](#). If you are a new user of Ground Truth, see [Output Data \(p. 404\)](#) to learn more about the Ground Truth output data format.

## Video Frame Labeling Job Overview

Use this page to learn about the object detection and object tracking video frame labeling jobs. The information on this page applies to both of these built-in task types.

The video frame labeling job is unique because of the following:

- You can either provide data objects that are ready to be annotated (video frames), or you can provide video files and have Ground Truth automatically extract video frames.
- Workers have the ability to save work as they go.
- You cannot use the Amazon Mechanical Turk workforce to complete your labeling tasks.
- Ground Truth provides a worker UI, as well as assistive and basic labeling tools, to help workers complete your tasks. You do not need to provide a worker task template.

Use the following topics to learn more.

#### Topics

- [Input Data \(p. 211\)](#)
- [Job Completion Times \(p. 211\)](#)
- [Task Types \(p. 211\)](#)
- [Workforces \(p. 212\)](#)
- [Worker User Interface \(UI\) \(p. 212\)](#)
- [Video Frame Job Permission Requirements \(p. 213\)](#)

### [Input Data](#)

The video frame labeling job uses *sequences* of video frames. A single sequence is a series of images that have been extracted from a single video. You can either provide your own sequences of video frames, or have Ground Truth automatically extract video frame sequences from your video files. To learn more, see [Provide Video Files \(p. 400\)](#).

Ground Truth uses sequence files to identify all images in a single sequence. All of the sequences that you want to include in a single labeling job are identified in an input manifest file. Each sequence is used to create a single worker task. You can automatically create sequence files and an input manifest file using Ground Truth automatic data setup. To learn more, see [Automated Video Frame Input Data Setup \(p. 401\)](#).

To learn how to manually create sequence files and an input manifest file, see [Create a Video Frame Input Manifest File \(p. 402\)](#).

### [Job Completion Times](#)

Video and video frame labeling jobs can take workers hours to complete. You can set the total amount of time that workers can work on each task when you create a labeling job. The maximum time you can set for workers to work on tasks is 7 days. The default value is 3 days.

We strongly recommend that you create tasks that workers can complete within 12 hours. Workers must keep the worker UI open while working on a task. They can save work as they go and Ground Truth saves their work every 15 minutes.

When using the SageMaker `CreateLabelingJob` API operation, set the total time a task is available to workers in the `TaskTimeLimitInSeconds` parameter of `HumanTaskConfig`.

When you create a labeling job in the console, you can specify this time limit when you select your workforce type and your work team.

### [Task Types](#)

When you create a video object tracking or video object detection labeling job, you specify the type of annotation that you want workers to create while working on your labeling task. The annotation type determines the type of output data Ground Truth returns and defines the *task type* for your labeling job.

If you are creating a labeling job using the API operation [CreateLabelingJob](#), you specify the task type using the label category configuration file parameter `annotationType`. To learn more, see [Create a Labeling Category Configuration File with Label Category and Frame Attributes \(p. 348\)](#).

The following task types are available for both video object tracking or video object detection labeling jobs:

- **Bounding box** – Workers are provided with tools to create bounding box annotations. A bounding box is a box that a worker draws around an objects to identify the pixel-location and label of that object in the frame.
- **Polyline** – Workers are provided with tools to create polyline annotations. A polyline is defined by the series of ordered x, y coordinates. Each point added to the polyline is connected to the previous point by a line. The polyline does not have to be closed (the start point and end point do not have to be the same) and there are no restrictions on the angles formed between lines.
- **Polygon** – Workers are provided with tools to create polygon annotations. A polygon is a closed shape defined by a series of ordered x, y coordinates. Each point added to the polygon is connected to the previous point by a line and there are no restrictions on the angles formed between lines. Two lines (sides) of the polygon cannot cross. The start and end point of a polygon must be the same.
- **Keypoint** – Workers are provided with tools to create keypoint annotations. A keypoint is a single point associated with an x, y coordinate in the video frame.

## Workforces

When you create a video frame labeling job, you need to specify a work team to complete your annotation tasks. You can choose a work team from a private workforce of your own workers, or from a vendor workforce that you select in the Amazon Web Services Marketplace. You cannot use the Amazon Mechanical Turk workforce for video frame labeling jobs.

To learn more about vendor workforces, see [Managing Vendor Workforces \(p. 460\)](#).

To learn how to create and manage a private workforce, see [Use a Private Workforce \(p. 461\)](#).

## Worker User Interface (UI)

Ground Truth provides a worker user interface (UI), tools, and assistive labeling features to help workers complete your video labeling tasks. You can preview the worker UI when you create a labeling job in the console.

When you create a labeling job using the API operation [CreateLabelingJob](#), you must provide an ARN provided by Ground Truth in the parameter `HumanTaskUiArn` to specify the worker UI for your task type. You can use `HumanTaskUiArn` with the SageMaker [RenderUiTemplate](#) API operation to preview the worker UI.

You provide worker instructions, labels, and optionally, attributes that workers can use to provide more information about labels and video frames. These attributes are referred to as label category attributes and frame attributes respectively. They are all displayed in the worker UI.

## Label Category and Frame Attributes

When you create a video object tracking or video object detection labeling job, you can add one or more *label category attributes* and *frame attributes*:

- **Label category attribute** – A list of options (strings), a free form text box, or a numeric field associated with one or more labels. It is used by workers to provide metadata about a label.
- **Frame attribute** – A list of options (strings), a free form text box, or a numeric field that appears on each video frame a worker is sent to annotate. It is used by workers to provide metadata about video frames.

Additionally, you can use label and frame attributes to have workers verify labels in a video frame label verification job.

Use the following sections to learn more about these attributes. To learn how to add label category and frame attributes to a labeling job, use the **Create Labeling Job** sections on the [task type page \(p. 202\)](#) of your choice.

### Frame level Attributes

Add frame attributes to give workers the ability to provide more information about individual video frames. Each frame attribute you add appears on all frames.

For example, you can add a number-frame attribute to have workers identify the number of objects they see in a particular frame.

In another example, you may want to provide a free-form text box to give workers the ability to provide an answer to a question.

When you create a label verification job, you can add one or more frame attributes to ask workers to provide feedback on all labels in a video frame.

### Worker Instructions

You can provide worker instructions to help your workers complete your video frame labeling tasks. You might want to cover the following topics when writing your instructions:

- Best practices and things to avoid when annotating objects.
- The label category attributes provided (for object detection and object tracking tasks) and how to use them.
- How to save time while labeling by using keyboard shortcuts.

You can add your worker instructions using the SageMaker console while creating a labeling job. If you create a labeling job using the API operation `CreateLabelingJob`, you specify worker instructions in your label category configuration file.

In addition to your instructions, Ground Truth provides a link to help workers navigate and use the worker portal. View these instructions by selecting the task type on [Worker Instructions \(p. 214\)](#).

### Declining Tasks

Workers are able to decline tasks.

Workers decline a task if the instructions are not clear, input data is not displaying correctly, or if they encounter some other issue with the task. If the number of workers per dataset object (`NumberOfHumanWorkersPerDataObject`) decline the task, the data object is marked as expired and will not be sent to additional workers.

### Video Frame Job Permission Requirements

When you create a video frame labeling job, in addition to the permission requirements found in [Assign IAM Permissions to Use Ground Truth \(p. 440\)](#), you must add a CORS policy to your S3 bucket that contains your input manifest file.

### Add a CORS Permission Policy to S3 Bucket

When you create a video frame labeling job, you specify buckets in S3 where your input data and manifest file are located and where your output data will be stored. These buckets may be the same. You must attach the following Cross-origin resource sharing (CORS) policy to your input and output buckets. If you use the Amazon S3 console to add the policy to your bucket, you must use the JSON format.

## JSON

```
[  
  {  
    "AllowedHeaders": [  
      "*"  
    ],  
    "AllowedMethods": [  
      "GET",  
      "HEAD",  
      "PUT"  
    ],  
    "AllowedOrigins": [  
      "*"  
    ],  
    "ExposeHeaders": [  
      "Access-Control-Allow-Origin"  
    ],  
    "MaxAgeSeconds": 3000  
  }  
]
```

## XML

```
<?xml version="1.0" encoding="UTF-8"?>  
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">  
<CORSRule>  
  <AllowedOrigin>*</AllowedOrigin>  
  <AllowedMethod>GET</AllowedMethod>  
  <AllowedMethod>HEAD</AllowedMethod>  
  <AllowedMethod>PUT</AllowedMethod>  
  <MaxAgeSeconds>3000</MaxAgeSeconds>  
  <ExposeHeader>Access-Control-Allow-Origin</ExposeHeader>  
  <AllowedHeader>*</AllowedHeader>  
</CORSRule>  
</CORSConfiguration>
```

To learn how to add a CORS policy to an S3 bucket, see [How do I add cross-domain resource sharing with CORS?](#) in the Amazon Simple Storage Service Console User Guide.

## Worker Instructions

This topic provides an overview of the Ground Truth worker portal and the tools available to complete your video frame labeling task. First, select the type of task you are working on from **Topics**.

### Important

It is recommended that you complete your task using a Google Chrome or Firefox web browser.

For adjustment jobs, select the original labeling job task type that produced the labels you are adjusting. Review and adjust the labels in your task as needed.

### Topics

- [Work on Video Frame Object Tracking Tasks \(p. 214\)](#)
- [Work on Video Frame Object Detection Tasks \(p. 222\)](#)

## Work on Video Frame Object Tracking Tasks

Video frame object tracking tasks require you to track the movement of objects across video frames. A video frame is a still image from a video scene.

You can use the worker UI to navigate between video frames and use the tools provided to identify unique objects and track their movement from one frame to the next. Use this page to learn how to navigate your worker UI, use the tools provided, and complete your task.

It is recommended that you complete your task using a Google Chrome or Firefox web browser.

**Important**

If you see annotations have already been added to one or more video frames when you open your task, adjust those annotations and add additional annotations as needed.

**Topics**

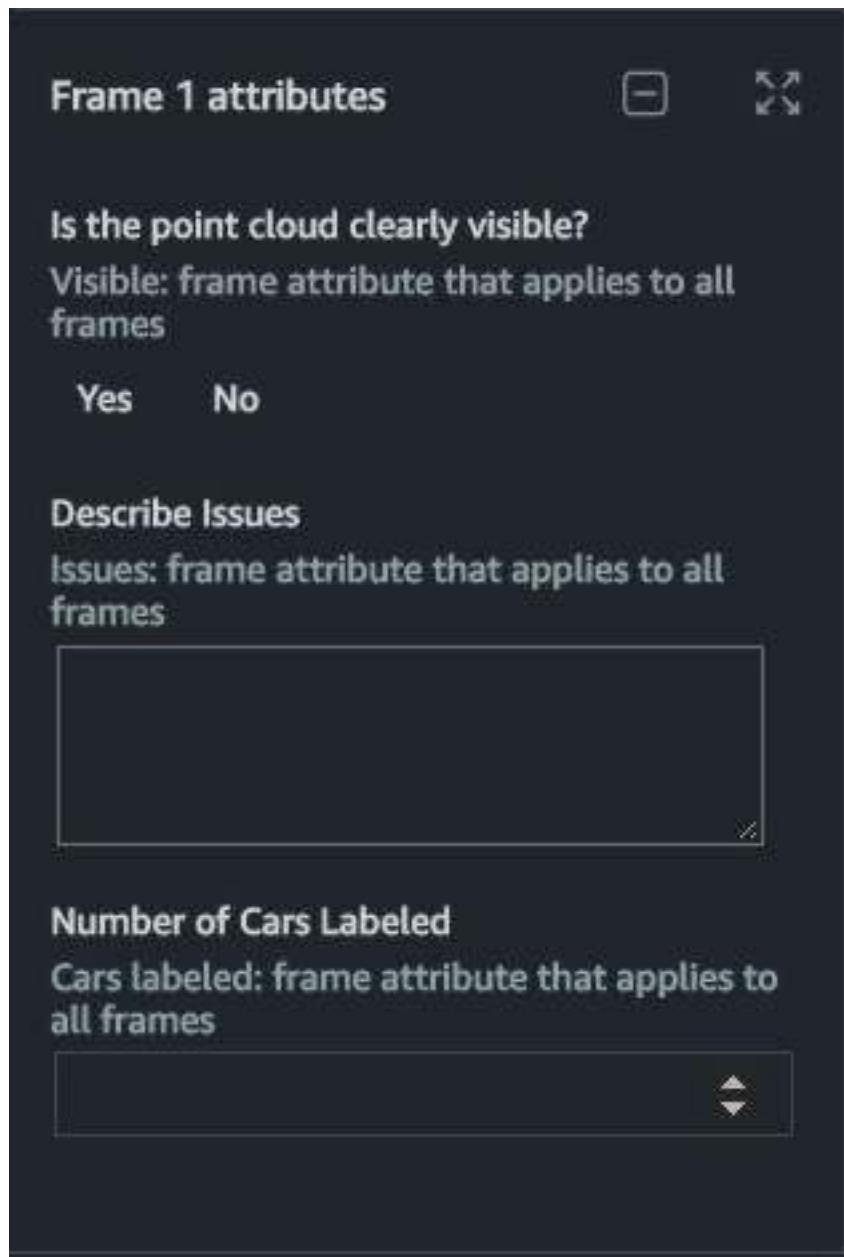
- [Your Task \(p. 215\)](#)
- [Navigate the UI \(p. 217\)](#)
- [Bulk Edit Label and Frame Attributes \(p. 217\)](#)
- [Tool Guide \(p. 218\)](#)
- [Icons Guide \(p. 220\)](#)
- [Shortcuts \(p. 221\)](#)
- [Release, Stop and Resume, and Decline Tasks \(p. 222\)](#)
- [Saving Your Work and Submitting \(p. 222\)](#)

**Your Task**

When you work on a video frame object tracking task, you need to select a category from the **Label category** menu on the right side of your worker portal to start annotating. After you've chosen a category, use the tools provided to annotate the objects that the category applies to. This annotation will be associated with a unique label ID that should only be used for that object. Use this same label ID to create additional annotations for the same object in all of the video frames that it appears in. Refer to [Tool Guide \(p. 218\)](#) to learn more about the tools provided.

After you've added a label, you may see a downward pointing arrow next to the label in the **Labels** menu. Select this arrow and then select one option for each label attribute you see to provide more information about that label.

You may see frame attributes under the **Labels** menu. These attributes will appear on each frame in your task. Use these attribute prompts to enter additional information about each frame.



After you've added a label, you can quickly add and edit a label category attribute value by using the downward pointing arrow next to the label in the **Labels** menu. If you select the pencil icon next to the label in the **Labels** menu, the **Edit instance** menu will appear. You can edit the label ID, label category, and label category attributes using this menu.

To edit an annotation, select the label of the annotation that you want to edit in the **Labels** menu or select the annotation in the frame. When you edit or delete an annotation, the action will only modify the annotation in a single frame.

If you are working on a task that includes a bounding box tool, use the predict next icon to predict the location of all bounding boxes that you have drawn in a frame in the next frame. If you select a single box and then select the predict next icon, only that box will be predicted in the next frame. If you have not added any boxes to the current frame, you will receive an error. You must add at least one box to the frame before using this feature.

After you've used the predict next icon, review the location of each box in the next frame and make adjustments to the box location and size if necessary.

For all other tools, you can use the **Copy to next** and **Copy to all** tools to copy your annotations to the next or all frames respectively.

### Navigate the UI

You can navigate between video frames using the navigation bar in the bottom-left corner of your UI.

Use the play button to automatically move through the entire sequence of frames.

Use the next frame and previous frame buttons to move forward or back one frame at a time. You can also input a frame number to navigate to that frame.

You can zoom in to and out of all video frames. Once you have zoomed into a video frame, you can move around in that frame using the move icon. When you set a new view in a single video frame by zooming and moving within that frame, all video frames are set to the same view. You can reset all video frames to their original view using the fit screen icon. For additional view options, see [Icons Guide \(p. 220\)](#).

When you are in the worker UI, you see the following menus:

- **Instructions** – Review these instructions before starting your task. Additionally, select **More instructions** and review these instructions.
- **Shortcuts** – Use this menu to view keyboard shortcuts that you can use to navigate video frames and use the tools provided.
- **Help** – Use this option to refer back to this documentation.

### Bulk Edit Label and Frame Attributes

You can bulk edit label attributes and frame attributes (attributes).

When you bulk edit an attribute, you specify one or more ranges of frames that you want to apply the edit to. The attribute you select is edited in all frames in that range, including the start and end frames you specify. When you bulk edit label attributes, the range you specify *must* contain the label that the label attribute is attached to. If you specify frames that do not contain this label, you will receive an error.

To bulk edit an attribute you *must* specify the desired value for the attribute first. For example, if you want to change an attribute from *Yes* to *No*, you must select *No*, and then perform the bulk edit.

You can also specify a new value for an attribute that has not been filled in and then use the bulk edit feature to fill in that value in multiple frames. To do this, select the desired value for the attribute and complete the following procedure.

#### To bulk edit a label or attribute:

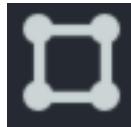
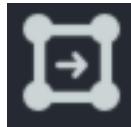
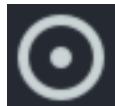
1. Use your mouse to right click the attribute you want to bulk edit.
2. Specify the range of frames you want to apply the bulk edit to using a dash (-) in the text box. For example, if you want to apply the edit to frames one through ten, enter 1-10. If you want to apply the edit to frames two to five, eight to ten and twenty enter 2-5, 8-10, 20.
3. Select **Confirm**.

If you get an error message, verify that you entered a valid range and that the label associated with the label attribute you are editing (if applicable) exists in all frames specified.

You can quickly add a label to all previous or subsequent frames using the **Duplicate to previous frames** and **Duplicate to next frames** options in the **Label** menu at the top of your screen.

## Tool Guide

Your task will include one or more tools. The tool provided dictates the type of annotations you will create to identify and track objects. Use the following table to learn more about each tool provided.

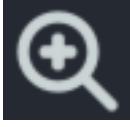
Tool	Icon	Action	Description
Bounding box		Add a bounding box annotation.	Choose this icon to add a bounding box. Each bounding box you add is associated with the category you choose from the Label category drop down menu. Select the bounding box or its associated label to adjust it.
Bounding box		Predict bounding boxes in the next frame.	Select a bounding box, and then choose this icon to predict the location of that box in the next frame. You can select the icon multiple times in a row to automatically detect the location of box in multiple frames. For example, select this icon 5 times to predict the location of a bounding box in the next 5 frames.
Keypoint		Add a keypoint annotation.	Choose this icon to add a keypoint. Click on an object in the image to place the keypoint at that location.  Each keypoint you add is associated with the category you choose from the Label category drop down menu. Select a keypoint or its associated label to adjust it.
Polyline		Add a polyline annotation.	Choose this icon to add a polyline. To add a polyline, continuously click around the object of interest to add new points. To stop drawing a polyline, select the last point that you placed a second time

Tool	Icon	Action	Description
			<p>(this point will be green), or press <b>Enter</b> on your keyboard.</p> <p>Each point added to the polyline is connected to the previous point by a line. The polyline does not have to be closed (the start point and end point do not have to be the same) and there are no restrictions on the angles formed between lines.</p> <p>Each polyline you add is associated with the category you choose from the Label category drop down menu. Select the polyline or its associated label to adjust it.</p>
Polygon		<p>Add a polygon annotation.</p>	<p>Choose this icon to add a polygon. To add a polygon, continuously click around the object of interest to add new points. To stop drawing the polygon, select the start point (this point will be green).</p> <p>A polygon is a closed shape defined by a series of points that you place. Each point added to the polygon is connected to the previous point by a line and there are no restrictions on the angles formed between lines. The start and end point must be the same.</p> <p>Each polyline you add is associated with the category you choose from the Label category drop down menu. Select the polyline or its associated label to adjust it.</p>

Tool	Icon	Action	Description
Copy to Next		Copy annotations to the next frame.	If one or more annotations are selected in the current frame, those annotations are copied to the next frame. If no annotations are selected, all annotations in the current frame will be copied to the next frame.
Copy to All		Copy annotations to all subsequent frames.	If one or more annotations are selected in the current frame, those annotations are copied to all subsequent frames. If no annotations are selected, all annotations in the current frame will be copied to all subsequent frames.

### Icons Guide

Use this table to learn about the icons you see in your UI. You can automatically select some of these icons using the keyboard shortcuts found in the **Shortcuts** menu.

Icon	Action	Description
	brightness	Choose this icon to adjust the brightness of all video frames.
	contrast	Choose this icon to adjust the contrast of all video frames.
	zoom in	Choose this icon to zoom into all of the video frames.

Icon	Action	Description
	zoom out	Choose this icon to zoom out of all of the video frames.
	move screen	After you've zoomed into a video frame, choose this icon to move around in that video frame. You can move around the video frame using your mouse by clicking and dragging the frame in the direction you want it to move. This will change the view in all view frames.
	fit screen	Reset all video frames to their original position.
	undo	Undo an action. You can use this icon to remove a bounding box that you just added, or to undo an adjustment you made to a bounding box.
	redo	Redo an action that was undone using the undo icon.
	delete label	Delete a label. This will delete the bounding box associated with the label in a single frame.
	show or hide label	Select this icon to show a label that has been hidden. If this icon has a slash through it, select it to hide the label.
	edit label	Select this icon to open the <b>Edit instance</b> menu. Use this menu to edit a label category, ID, and to add or edit label attributes.

## Shortcuts

The keyboard shortcuts listed in the **Shortcuts** menu can help you quickly select icons, undo and redo annotations, and use tools to add and edit annotations. For example, once you add a bounding box, you can use **P** to quickly predict the location of that box in subsequent frames.

Before you start your task, it is recommended that you review the **Shortcuts** menu and become acquainted with these commands.

## Release, Stop and Resume, and Decline Tasks

When you open the labeling task, three buttons on the top right allow you to decline the task (**Decline task**), release it (**Release task**), and stop and resume it at a later time (**Stop and resume later**). The following list describes what happens when you select one of these options:

- **Decline task:** You should only decline a task if something is wrong with the task, such as unclear video frame images or an issue with the UI. If you decline a task, you will not be able to return to the task.
- **Release Task:** Use this option to release a task and allow others to work on it. When you release a task, you loose all work done on that task and other workers on your team can pick it up. If enough workers pick up the task, you may not be able to return to it. When you select this button and then select **Confirm**, you are returned to the worker portal. If the task is still available, its status will be **Available**. If other workers pick it up, it will disappear from your portal.
- **Stop and resume later:** You can use the **Stop and resume later** button to stop working and return to the task at a later time. You should use the **Save** button to save your work before you select **Stop and resume later**. When you select this button and then select **Confirm**, you are returned to the worker portal, and the task status is **Stopped**. You can select the same task to resume work on it.

Be aware that the person that creates your labeling tasks specifies a time limit in which all tasks must be completed by. If you do not return to and complete this task within that time limit, it will expire and your work will not be submitted. Contact your administrator for more information.

## Saving Your Work and Submitting

You should periodically save your work using the **Save** button. Ground Truth will automatically save your work every 15 minutes.

When you open a task, you must complete your work on it before pressing **Submit**.

## Work on Video Frame Object Detection Tasks

Video frame object detection tasks required you to classify and identify the location of objects in video frames using annotations. A video frame is a still image from a video scene.

You can use the worker UI to navigate between video frames and create annotations to identify objects of interest. Use the sections on this page to learn how to navigate your worker UI, use the tools provided, and complete your task.

It is recommended that you complete your task using a Google Chrome web browser.

### Important

If you see annotations have already been added to one or more video frames when you open your task, adjust those annotations and add additional annotations as needed.

### Topics

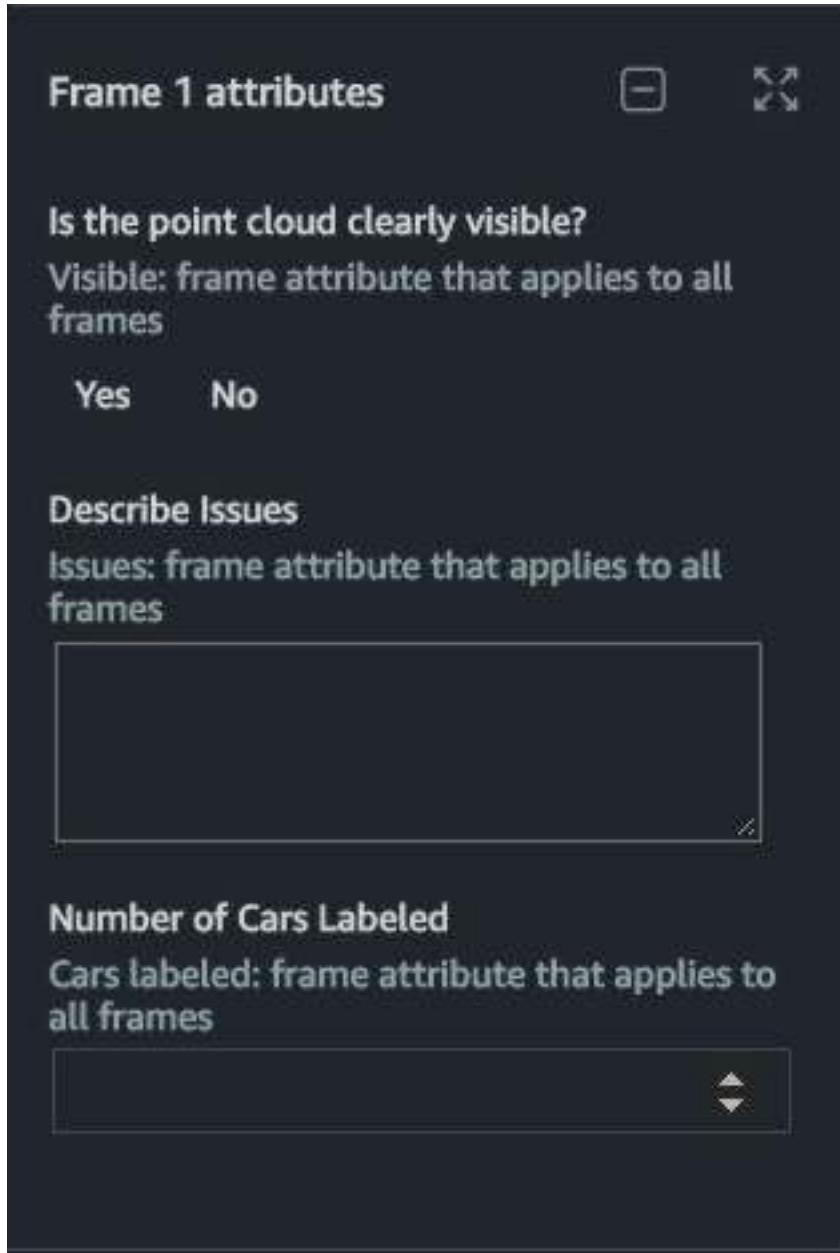
- [Your Task \(p. 223\)](#)
- [Navigate the UI \(p. 224\)](#)
- [Bulk Edit Label and Frame Attributes \(p. 224\)](#)
- [Tool Guide \(p. 225\)](#)
- [UI Icon Guide \(p. 228\)](#)
- [Shortcuts \(p. 229\)](#)
- [Release, Stop and Resume, and Decline Tasks \(p. 229\)](#)
- [Saving Your Work and Submitting \(p. 230\)](#)

## Your Task

When you work on a video frame object detection task, you need to select a category from the **Label category** menu on the right side of your worker portal to start annotating. After you've chosen a category, draw annotations around objects that this category applies to. To learn more about the tools you see in your worker UI, refer to the [Tool Guide \(p. 225\)](#).

After you've added a label, you may see a downward pointing arrow next to the label in the **Labels** menu. Select this arrow and then select one option for each label attribute you see to provide more information about that label.

You may see frame attributes under the **Labels** menu. These attributes will appear on each frame in your task. Use these attribute prompts to enter additional information about each frame.



To edit an annotation, select the label of the annotation that you want to edit in the **Labels** menu or select the annotation in the frame. When you edit or delete an annotation, the action will only modify the annotation in a single frame.

If you are working on a task that includes a bounding box tool, use the predict next icon to predict the location of all bounding boxes that you have drawn in a frame in the next frame. If you select a single box and then select the predict next icon, only that box will be predicted in the next frame. If you have not added any boxes to the current frame, you will receive an error. You must add at least one box to the frame before using this feature.

#### Note

The predict next feature will not overwrite manually created annotations. It will only add annotations. If you use predict next and as a result have more than one bounding box around a single object, delete all but one box. Each object should only be identified with a single box.

After you've used the predict next icon, review the location of each box in the next frame and make adjustments to the box location and size if necessary.

For all other tools, you can use the **Copy to next** and **Copy to all** tools to copy your annotations to the next or all frames respectively.

### Navigate the UI

You can navigate between video frames using the navigation bar in the bottom-left corner of your UI.

Use the play button to automatically play through multiple frames.

Use the next frame and previous frame buttons to move forward or back one frame at a time. You can also input a frame number to navigate to that frame.

You can zoom in to and out of all video frames. Once you have zoomed into a video frame, you can move around in that frame using the move icon. When you navigate to a new view in a single video frame by zooming and moving within that frame, all video frames are set to the same view. You can reset all video frames to their original view using the fit screen icon. To learn more, see [UI Icon Guide \(p. 228\)](#).

When you are in the worker UI, you see the following menus:

- **Instructions** – Review these instructions before starting your task. Additionally, select **More instructions** and review these instructions.
- **Shortcuts** – Use this menu to view keyboard shortcuts that you can use to navigate video frames and use the annotation tools provided.
- **Help** – Use this option to refer back to this documentation.

If you

### Bulk Edit Label and Frame Attributes

You can bulk edit label attributes and frame attributes (attributes).

When you bulk edit an attribute, you specify one or more ranges of frames that you want to apply the edit to. The attribute you select is edited in all frames in that range, including the start and end frames you specify. When you bulk edit label attributes, the range you specify *must* contain the label that the label attribute is attached to. If you specify frames that do not contain this label, you will receive an error.

To bulk edit an attribute you *must* specify the desired value for the attribute first. For example, if you want to change an attribute from *Yes* to *No*, you must select *No*, and then perform the bulk edit.

You can also specify a new value for an attribute that has not been filled in and then use the bulk edit feature to fill in that value in multiple frames. To do this, select the desired value for the attribute and complete the following procedure.

### To bulk edit a label or attribute:

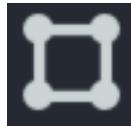
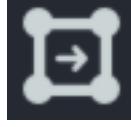
1. Use your mouse to right click the attribute you want to bulk edit.
2. Specify the range of frames you want to apply the bulk edit to using a dash (-) in the text box. For example, if you want to apply the edit to frames one through ten, enter 1-10. If you want to apply the edit to frames two to five, eight to ten and twenty enter 2-5, 8-10, 20.
3. Select **Confirm**.

If you get an error message, verify that you entered a valid range and that the label associated with the label attribute you are editing (if applicable) exists in all frames specified.

You can quickly add a label to all previous or subsequent frames using the **Duplicate to previous frames** and **Duplicate to next frames** options in the **Label** menu at the top of your screen.

### Tool Guide

Your task will include one or more tools. The tool provided dictates the type of annotations you will create to identify and label objects. Use the following table to learn more about the tool or tools you may see in your worker UI.

Tool	Icon	Action	Description
Bounding box		Add a bounding box annotation.	Choose this icon to add a bounding box. Each bounding box you add is associated with the category you choose from the Label category drop down menu. Select the bounding box or its associated label to adjust it.
Predict next		Predict bounding boxes in the next frame.	Select a bounding box, and then choose this icon to predict the location of that box in the next frame. You can select the icon multiple times in a row to automatically detect the location of box in multiple frames. For example, select this icon 5 times to predict the location of a bounding box in the next 5 frames.
Keypoint		Add a keypoint annotation.	Choose this icon to add a keypoint. Click on an object the image to place the keypoint at that location.  Each keypoint you add is associated with the

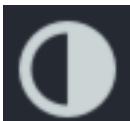
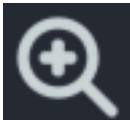
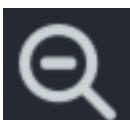
Tool	Icon	Action	Description
			category you choose from the Label category drop down menu. Select a keypoint or its associated label to adjust it.
Polyline		Add a polyline annotation.	<p>Choose this icon to add a polyline. To add a polyline, continuously click around the object of interest to add new points. To stop drawing a polyline, select the last point that you placed a second time (this point will be green), or press <b>Enter</b> on your keyboard.</p> <p>Each point added to the polyline is connected to the previous point by a line. The polyline does not have to be closed (the start point and end point do not have to be the same) and there are no restrictions on the angles formed between lines.</p> <p>Each polyline you add is associated with the category you choose from the Label category drop down menu. Select the polyline or its associated label to adjust it.</p>

Tool	Icon	Action	Description
Polygon		<p>Add a polygon annotation.</p> <p>A polygon is a closed shape defined by a series of points that you place. Each point added to the polygon is connected to the previous point by a line and there are no restrictions on the angles formed between lines. Two lines (sides) of the polygon cannot cross. A line will become red if it violates this condition. The start and end point must be the same.</p> <p>Each polyline you add is associated with the category you choose from the Label category drop down menu. Select the poly</p>	<p>Choose this icon to add a polygon. To add a polygon, continuously click around the object of interest to add new points. To stop drawing the polygon, select the start point (this point will be green).</p>
Copy to Next		Copy annotations to the next frame.	If one or more annotations are selected in the current frame, those annotations are copied to the next frame. If no annotations are selected, all annotations in the current frame will be copied to the next frame.

Tool	Icon	Action	Description
Copy to All		Copy annotations to all subsequent frames.	If one or more annotations are selected in the current frame, those annotations are copied to all subsequent frames. If no annotations are selected, all annotations in the current frame will be copied to all subsequent frames.

### UI Icon Guide

Use this table to learn about the icons you see in your worker task portal. You can automatically select these icons using the keyboard shortcuts found in the **Shortcuts** menu.

Icon		Description
	brightness	Choose this icon to adjust the brightness of all video frames.
	contrast	Choose this icon to adjust the contrast of all video frames.
	zoom in	Choose this icon to zoom into all of the video frames.
	zoom out	Choose this icon to zoom out of all of the video frames.
	move screen	After you've zoomed into a video frame, choose this icon to move around in that video frame. You can move around in the video frame using your mouse by clicking and dragging the frame in the direction you want it to move. This will change the view in all view frames.
	fit screen	Reset all video frames to their original position.

Icon		Description
	undo	Undo an action. You can use this icon to remove a bounding box that you just added, or to undo an adjustment you made to a bounding box.
	redo	Redo an action that was undone using the undo icon.
	delete label	Delete a label. This will delete the bounding box associated with the label in a single frame.
	show or hide label	Select this icon to show a label that has been hidden. If this icon has a slash through it, select it to hide the label.

## Shortcuts

The keyboard shortcuts listed in the **Shortcuts** menu can help you quickly select icons, undo and redo annotations, and use tools to add and edit annotations. For example, once you add a bounding box, you can use **P** to quickly predict the location of that box in subsequent frames.

Before you start your task, it is recommended that you review the **Shortcuts** menu and become acquainted with these commands.

## Release, Stop and Resume, and Decline Tasks

When you open the labeling task, three buttons on the top right allow you to decline the task (**Decline task**), release it (**Release task**), and stop and resume it at a later time (**Stop and resume later**). The following list describes what happens when you select one of these options:

- **Decline task:** You should only decline a task if something is wrong with the task, such as unclear video frame images or an issue with the UI. If you decline a task, you will not be able to return to the task.
- **Release Task:** Use this option to release a task and allow others to work on it. When you release a task, you lose all work done on that task and other workers on your team can pick it up. If enough workers pick up the task, you may not be able to return to it. When you select this button and then select **Confirm**, you are returned to the worker portal. If the task is still available, its status will be **Available**. If other workers pick it up, it will disappear from your portal.
- **Stop and resume later:** You can use the **Stop and resume later** button to stop working and return to the task at a later time. You should use the **Save** button to save your work before you select **Stop and resume later**. When you select this button and then select **Confirm**, you are returned to the worker portal, and the task status is **Stopped**. You can select the same task to resume work on it.

Be aware that the person that creates your labeling tasks specifies a time limit in which all tasks must be completed by. If you do not return to and complete this task within that time limit, it will expire and your work will not be submitted. Contact your administrator for more information.

### Saving Your Work and Submitting

You should periodically save your work. Ground Truth automatically saves your work every 15 minutes.

When you open a task, you must complete your work before pressing **Submit**.

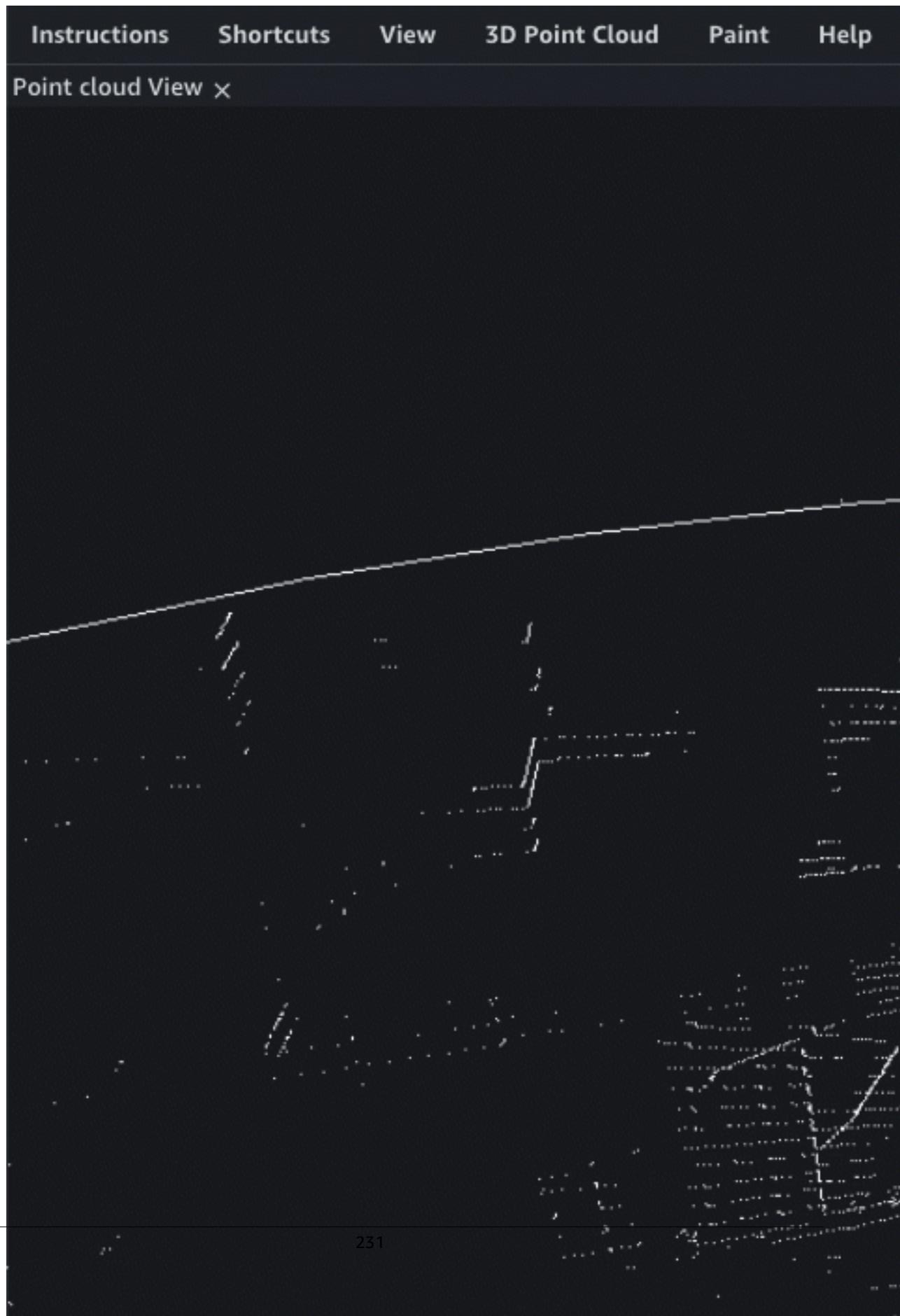
## Use Ground Truth to Label 3D Point Clouds

Create a 3D point cloud labeling job to have workers label objects in 3D point clouds generated from 3D sensors like Light Detection and Ranging (LiDAR) sensors and depth cameras, or generated from 3D reconstruction by stitching images captured by an agent like a drone.

### 3D Point Clouds

Point clouds are made up of three-dimensional (3D) visual data that consists of points. Each point is described using three coordinates, typically *x*, *y*, and *z*. To add color or variations in point intensity to the point cloud, points may be described with additional attributes, such as *i* for intensity or values for the red (*r*), green (*g*), and blue (*b*) 8-bit color channels. When you create a Ground Truth 3D point cloud labeling job, you can provide point cloud and, optionally, sensor fusion data.

The following image shows a single, 3D point cloud scene rendered by Ground Truth and displayed in the semantic segmentation worker UI.



## LiDAR

A Light Detection and Ranging (LiDAR) sensor is a common type of sensor used to collect measurements that are used to generate point cloud data. LiDAR is a remote sensing method that uses light in the form of a pulsed laser to measure the distances of objects from the sensor. You can provide 3D point cloud data generated from a LiDAR sensor for a Ground Truth 3D point cloud labeling job using the raw data formats described in [Accepted Raw 3D Data Formats \(p. 375\)](#).

## Sensor Fusion

Ground Truth 3D point cloud labeling jobs include a sensor fusion feature that supports video camera sensor fusion for all task types. Some sensors come with multiple LiDAR devices and video cameras that capture images and associate them with a LiDAR frame. To help annotators visually complete your tasks with high confidence, you can use the Ground Truth sensor fusion feature to project annotations (labels) from a 3D point cloud to 2D camera images and vice versa using 3D scanner (such as LiDAR) extrinsic matrix and camera extrinsic and intrinsic matrices. To learn more, see [Sensor Fusion \(p. 391\)](#).

## Label 3D Point Clouds

Ground Truth provides a user interface (UI) and tools that workers use to label or *annotate* 3D point clouds. When you use the object detection or semantic segmentation task types, workers can annotate a single point cloud frame. When you use object tracking, workers annotate a sequence of frames. You can use object tracking to track object movement across all frames in a sequence.

The following demonstrates how a worker would use the Ground Truth worker portal and tools to annotate a 3D point cloud for an object detection task. For similar visual examples of other task types, see [3D Point Cloud Task types \(p. 234\)](#).

Hello, chopt@amazon.com

Instructions    **Shortcuts**    Label    View    3D Point Cloud    Help

**Shortcuts**    X    Point cloud View X

Double click on the point cloud to zoom in

Double click on the label ID to zoom in the object on point cloud.

**3D Cuboid Controls**

When you are in Edit Cuboid mode

C Create

V Edit

Cmd + Drag Change dimension

Option + Drag Move cuboid

Option + O Fit label to points

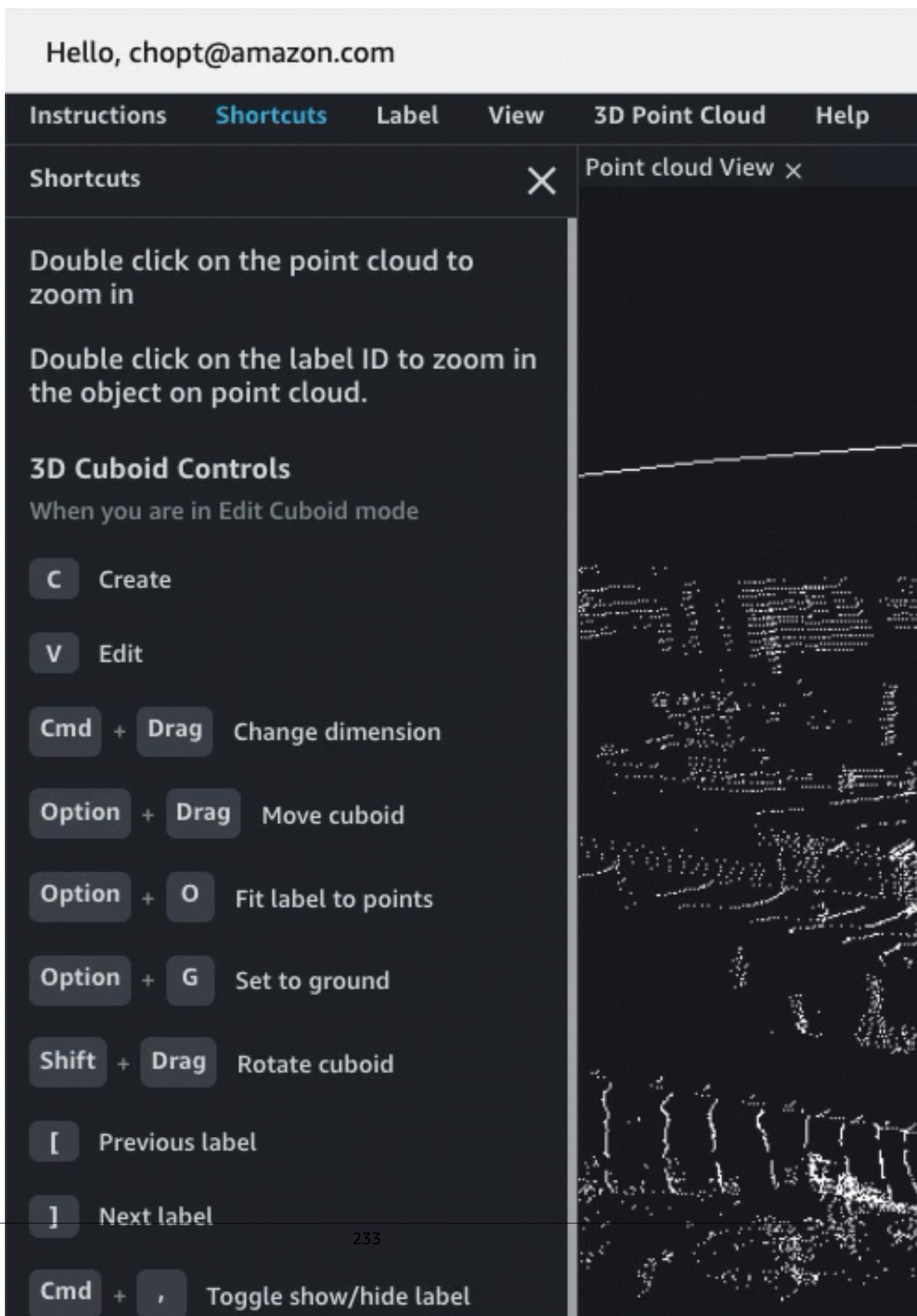
Option + G Set to ground

Shift + Drag Rotate cuboid

[ Previous label

] Next label

Cmd + , Toggle show/hide label



## Assistive Labeling Tools for Point Cloud Annotation

Ground Truth offers assistive labeling tools to help workers complete your point cloud annotation tasks faster and with more accuracy. For details about assistive labeling tools that are included in the worker UI for each task type, [select a task type \(p. 234\)](#) and refer to the [View the Worker Task Interface](#) section of that page.

## Next Steps

You can create six types of tasks when you use Ground Truth 3D point cloud labeling jobs. Use the topics in [3D Point Cloud Task types \(p. 234\)](#) to learn more about these *task types* and to learn how to create a labeling job using the task type of your choice.

The 3D point cloud labeling job is different from other Ground Truth labeling modalities. Before creating a labeling job, we recommend that you read [3D Point Cloud Labeling Jobs Overview \(p. 259\)](#). Additionally, review input data quotas in [3D Point Cloud and Video Frame Labeling Job Quotas \(p. 373\)](#).

For an end-to-end demo using the SageMaker API and Amazon Python SDK (boto 3) to create a 3D point cloud labeling job, see [create-3D-pointcloud-labeling-job.ipynb](#) in the [SageMaker Examples notebook tab](#).

### Important

If you use a notebook instance created before June 5th, 2020 to run this notebook, you must stop and restart that notebook instance for the notebook to work.

### Topics

- [3D Point Cloud Task types \(p. 234\)](#)
- [3D Point Cloud Labeling Jobs Overview \(p. 259\)](#)
- [Worker Instructions \(p. 262\)](#)

## 3D Point Cloud Task types

You can use Ground Truth 3D point cloud labeling modality for a variety of use cases. The following list briefly describes each 3D point cloud task type. For additional details and instructions on how to create a labeling job using a specific task type, select the task type name to see its task type page.

- **3D point cloud object detection** – Use this task type when you want workers to locate and classify objects in a 3D point cloud by adding and fitting 3D cuboids around objects.
- **3D point cloud object tracking** – Use this task type when you want workers to add and fit 3D cuboids around objects to track their movement across a sequence of 3D point cloud frames. For example, you can use this task type to ask workers to track the movement of vehicles across multiple point cloud frames.
- **3D point cloud semantic segmentation** – Use this task type when you want workers to create a point-level semantic segmentation mask by painting objects in a 3D point cloud using different colors where each color is assigned to one of the classes you specify.
- **3D point cloud adjustment task types** – Each of the task types above has an associated *adjustment* task type that you can use to audit and adjust annotations generated from a 3D point cloud labeling job. Refer to the task type page of the associated type to learn how to create an adjustment labeling job for that task.

### 3D Point Cloud Object Detection

Use this task type when you want workers to classify objects in a 3D point cloud by drawing 3D cuboids around objects. For example, you can use this task type to ask workers to identify different types of objects in a point cloud, such as cars, bikes, and pedestrians.

For this task type, the *data object* that workers label is a single point cloud frame. Ground Truth renders a 3D point cloud using point cloud data you provide. You can also provide camera data to give workers more visual information about scenes in the frame, and to help workers draw 3D cuboids around objects.

Ground Truth providers workers with tools to annotate objects with 9 degrees of movement ( $x,y,z,rx,ry,rz,l,w,h$ ) in three dimensions in both 3D scene and projected side views (top, side, and back). If you provide sensor fusion information (like camera data), when a worker adds a cuboid to identify an object in the 3D point cloud, the cuboid shows up and can be modified in the 2D images. After a cuboid has been added, all edits made to that cuboid in the 2D or 3D scene are projected into the other view.

You can create a job to adjust annotations created in a 3D point cloud object detection labeling job using the 3D point cloud object detection adjustment task type.

If you are a new user of the Ground Truth 3D point cloud labeling modality, we recommend you review [3D Point Cloud Labeling Jobs Overview \(p. 259\)](#). This labeling modality is different from other Ground Truth task types, and this page provides an overview of important details you should be aware of when creating a 3D point cloud labeling job.

### Topics

- [View the Worker Task Interface \(p. 235\)](#)
- [Create a 3D Point Cloud Object Detection Labeling Job \(p. 239\)](#)
- [Create a 3D Point Cloud Object Detection Adjustment or Verification Labeling Job \(p. 240\)](#)
- [Output Data Format \(p. 241\)](#)

### View the Worker Task Interface

Ground Truth provides workers with a web portal and tools to complete your 3D point cloud object detection annotation tasks. When you create the labeling job, you provide the Amazon Resource Name (ARN) for a pre-built Ground Truth worker UI in the `HumanTaskUiArn` parameter. When you create a labeling job using this task type in the console, this worker UI is automatically used. You can preview and interact with the worker UI when you create a labeling job in the console. If you are a new user, it is recommended that you create a labeling job using the console to ensure your label attributes, point cloud frames, and if applicable, images, appear as expected.

The following is a GIF of the 3D point cloud object detection worker task interface. If you provide camera data for sensor fusion in the world coordinate system, images are matched up with scenes in the point cloud frame. These images appear in the worker portal as shown in the following GIF.

Hello, chopt@amazon.com

Instructions    **Shortcuts**    Label    View    3D Point Cloud    Help

**Shortcuts**    X    Point cloud View X

Double click on the point cloud to zoom in

Double click on the label ID to zoom in the object on point cloud.

**3D Cuboid Controls**

When you are in Edit Cuboid mode

C Create

V Edit

Cmd + Drag Change dimension

Option + Drag Move cuboid

Option + O Fit label to points

Option + G Set to ground

Shift + Drag Rotate cuboid

[ Previous label

] Next label

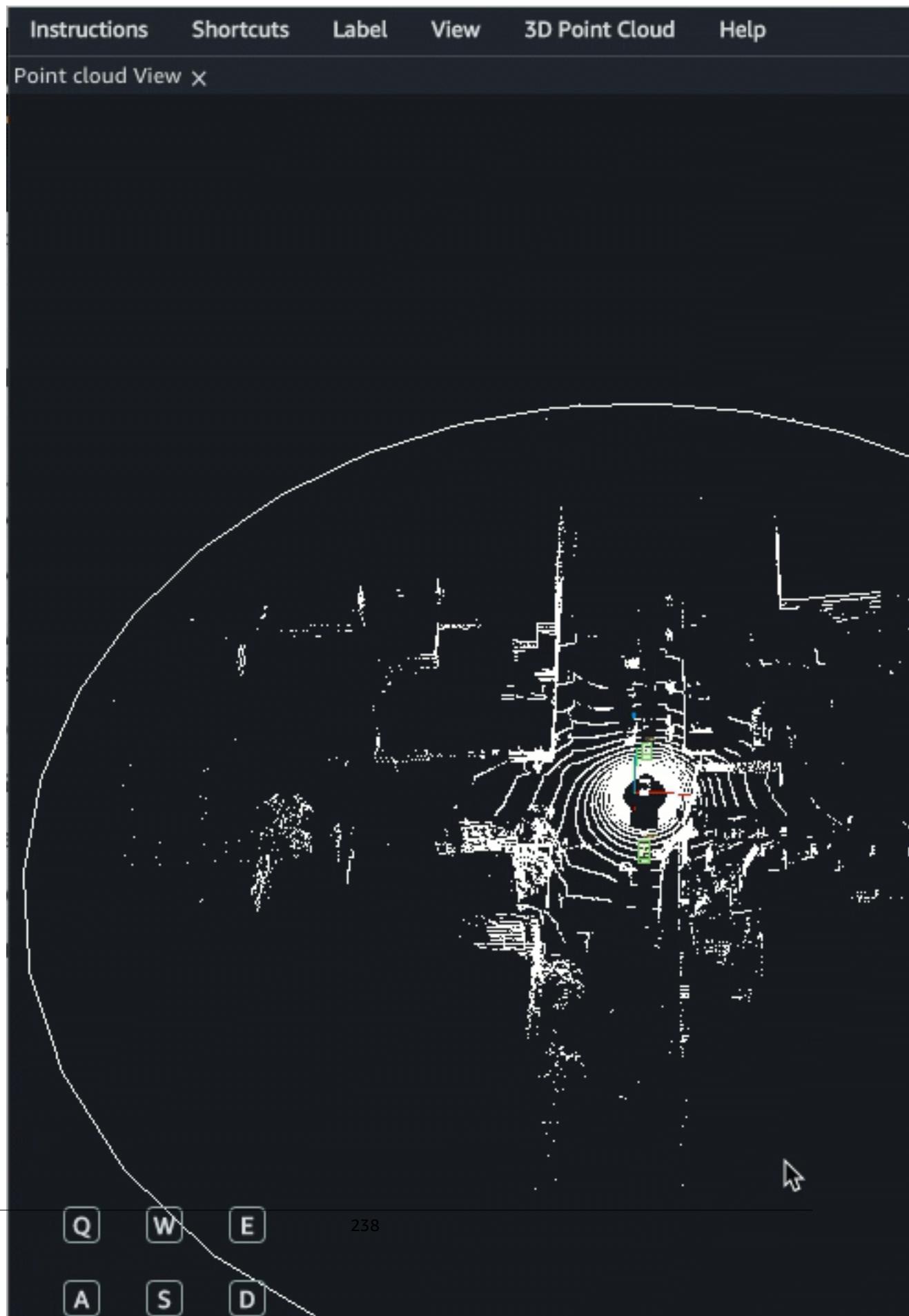
Cmd + , Toggle show/hide label

Worker can navigate in the 3D scene using their keyboard and mouse. They can:

- Double click on specific objects in the point cloud to zoom into them.
- Use a mouse-scroller or trackpad to zoom in and out of the point cloud.
- Use both keyboard arrow keys and Q, E, A, and D keys to move Up, Down, Left, Right. Use keyboard keys W and S to zoom in and out.

Once a worker places a cuboid in the 3D scene, a side-view will appear with the three projected side views: top, side, and back. These side-views show points in and around the placed cuboid and help workers refine cuboid boundaries in that area. Workers can zoom in and out of each of those side-views using their mouse.

The following video demonstrates movements around the 3D point cloud and in the side-view.



Additional view options and features are available in the **View** menu in the worker UI. See the [worker instruction page](#) for a comprehensive overview of the Worker UI.

### Assistive Labeling Tools

Ground Truth helps workers annotate 3D point clouds faster and more accurately using machine learning and computer vision powered assistive labeling tools for 3D point cloud object tracking tasks. The following assistive labeling tools are available for this task type:

- **Snapping** – Workers can add a cuboid around an object and use a keyboard shortcut or menu option to have Ground Truth's autofit tool snap the cuboid tightly around the object.
- **Set to ground** – After a worker adds a cuboid to the 3D scene, the worker can automatically snap the cuboid to the ground. For example, the worker can use this feature to snap a cuboid to the road or sidewalk in the scene.
- **Multi-view labeling** – After a worker adds a 3D cuboid to the 3D scene, a side panel displays front, side, and top perspectives to help the worker adjust the cuboid tightly around the object. In all of these views, the cuboid includes an arrow that indicates the orientation, or heading of the object. When the worker adjusts the cuboid, the adjustment will appear in real time on all of the views (that is, 3D, top, side, and front).
- **Sensor fusion** – If you provide data for sensor fusion, workers can adjust annotations in the 3D scenes and in 2D images, and the annotations will be projected into the other view in real time. Additionally, workers will have the option to view the direction the camera is facing and the camera frustum.
- **View options** – Enables workers to easily hide or view cuboids, label text, a ground mesh, and additional point attributes like color or intensity. Workers can also choose between perspective and orthogonal projections.

### Create a 3D Point Cloud Object Detection Labeling Job

You can create a 3D point cloud labeling job using the SageMaker console or API operation, [CreateLabelingJob](#). To create a labeling job for this task type you need the following:

- A single-frame input manifest file. To learn how to create this type of manifest file, see [Create a Point Cloud Frame Input Manifest File \(p. 376\)](#). If you are a new user of Ground Truth 3D point cloud labeling modalities, you may also want to review [Accepted Raw 3D Data Formats \(p. 375\)](#).
- A work team from a private or vendor workforce. You cannot use Amazon Mechanical Turk for video frame labeling jobs. To learn how to create workforces and work teams, see [Create and Manage Workforces \(p. 456\)](#).

Additionally, make sure that you have reviewed and satisfied the [Assign IAM Permissions to Use Ground Truth \(p. 440\)](#).

Use one of the following sections to learn how to create a labeling job using the console or an API.

#### Create a Labeling Job (Console)

You can follow the instructions [Create a Labeling Job \(Console\) \(p. 336\)](#) in order to learn how to create a 3D point cloud object detection labeling job in the SageMaker console. While you are creating your labeling job, be aware of the following:

- Your input manifest file must be a single-frame manifest file. For more information, see [Create a Point Cloud Frame Input Manifest File \(p. 376\)](#).
- Optionally, you can provide label category and frame attributes. Workers can assign one or more of these attributes to annotations to provide more information about that object. For example, you might want to use the attribute *occluded* to have workers identify when an object is partially obstructed.

- Automated data labeling and annotation consolidation are not supported for 3D point cloud labeling tasks.
- 3D point cloud object detection labeling jobs can take multiple hours to complete. You can specify a longer time limit for these labeling jobs when you select your work team (up to 7 days, or 604800 seconds).

## Create a Labeling Job (API)

This section covers details you need to know when you create a labeling job using the SageMaker API operation `CreateLabelingJob`. This API defines this operation for all Amazon SDKs. To see a list of language-specific SDKs supported for this operation, review the **See Also** section of `CreateLabelingJob`.

[Create a Labeling Job \(API\) \(p. 338\)](#), provides an overview of the `CreateLabelingJob` operation. Follow these instructions and do the following while you configure your request:

- You must enter an ARN for `HumanTaskUiArn`. Use `arn:aws:sagemaker:<region>:394669845002:human-task-ui/PointCloudObjectDetection`. Replace `<region>` with the Amazon Region you are creating the labeling job in.

There should not be an entry for the `UiTemplatesS3Uri` parameter.

- Your input manifest file must be a single-frame manifest file. For more information, see [Create a Point Cloud Frame Input Manifest File \(p. 376\)](#).
- You specify your labels, label category and frame attributes, and worker instructions in a label category configuration file. To learn how to create this file, see [Create a Labeling Category Configuration File with Label Category and Frame Attributes \(p. 348\)](#).
- You need to provide pre-defined ARNs for the pre-annotation and post-annotation (ACS) Lambda functions. These ARNs are specific to the Amazon Region you use to create your labeling job.
  - To find the pre-annotation Lambda ARN, refer to `PreHumanTaskLambdaArn`. Use the Region you are creating your labeling job in to find the correct ARN. For example, if you are creating your labeling job in us-east-1, the ARN will be `arn:aws:lambda:us-east-1:432418664414:function:PRE-3DPointCloudObjectDetection`.
  - To find the post-annotation Lambda ARN, refer to `AnnotationConsolidationLambdaArn`. Use the Region you are creating your labeling job in to find the correct ARN. For example, if you are creating your labeling job in us-east-1, the ARN will be `arn:aws:lambda:us-east-1:432418664414:function:ACS-3DPointCloudObjectDetection`.
- The number of workers specified in `NumberOfHumanWorkersPerDataObject` must be 1.
- Automated data labeling is not supported for 3D point cloud labeling jobs. You should not specify values for parameters in `LabelingJobAlgorithmsConfig`.
- 3D point cloud object detection labeling jobs can take multiple hours to complete. You can specify a longer time limit for these labeling jobs in `TaskTimeLimitInSeconds` (up to 7 days, or 604,800 seconds).

## Create a 3D Point Cloud Object Detection Adjustment or Verification Labeling Job

You can create an adjustment or verification labeling job using the Ground Truth console or `CreateLabelingJob` API. To learn more about adjustment and verification labeling jobs, and to learn how create one, see [Verify and Adjust Labels \(p. 293\)](#).

When you create an adjustment labeling job, your input data to the labeling job can include labels, and yaw, pitch, and roll measurements from a previous labeling job or external source. In the adjustment job, pitch, and roll will be visualized in the worker UI, but cannot be modified. Yaw is adjustable.

Ground Truth uses Tait-Bryan angles with the following intrinsic rotations to visualize yaw, pitch and roll in the worker UI. First, rotation is applied to the vehicle according to the z-axis (yaw). Next, the rotated vehicle is rotated according to the intrinsic y'-axis (pitch). Finally, the vehicle is rotated according to the intrinsic x''-axis (roll).

### Output Data Format

When you create a 3D point cloud object detection labeling job, tasks are sent to workers. When these workers complete their tasks, labels are written to the Amazon S3 bucket you specified when you created the labeling job. The output data format determines what you see in your Amazon S3 bucket when your labeling job status ([LabelingJobStatus](#)) is Completed.

If you are a new user of Ground Truth, see [Output Data \(p. 404\)](#) to learn more about the Ground Truth output data format. To learn about the 3D point cloud object detection output data format, see [3D Point Cloud Object Detection Output \(p. 421\)](#).

## 3D Point Cloud Object Tracking

Use this task type when you want workers to add and fit 3D cuboids around objects to track their movement across 3D point cloud frames. For example, you can use this task type to ask workers to track the movement of vehicles across multiple point cloud frames.

For this task type, the data object that workers label is a sequence of point cloud frames. A *sequence* is defined as a temporal series of point cloud frames. Ground Truth renders a series of 3D point cloud visualizations using a sequence you provide and workers can switch between these 3D point cloud frames in the worker task interface.

Ground Truth providers workers with tools to annotate objects with 9 degrees of rotation: (x,y,z,rx,ry,rz,l,w,h) in three dimensions in both 3D scene and projected side views (top, side, and back). When a worker draws a cuboid around an object, that cuboid is given a unique ID, for example `Car : 1` for one car in the sequence and `Car : 2` for another. Workers use that ID to label the same object in multiple frames.

You can also provide camera data to give workers more visual information about scenes in the frame, and to help workers draw 3D cuboids around objects. When a worker adds a 3D cuboid to identify an object in either the 2D image or the 3D point cloud, and the cuboid shows up in the other view.

You can adjust annotations created in a 3D point cloud object detection labeling job using the 3D point cloud object tracking adjustment task type.

If you are a new user of the Ground Truth 3D point cloud labeling modality, we recommend you review [3D Point Cloud Labeling Jobs Overview \(p. 259\)](#). This labeling modality is different from other Ground Truth task types, and this page provides an overview of important details you should be aware of when creating a 3D point cloud labeling job.

### Topics

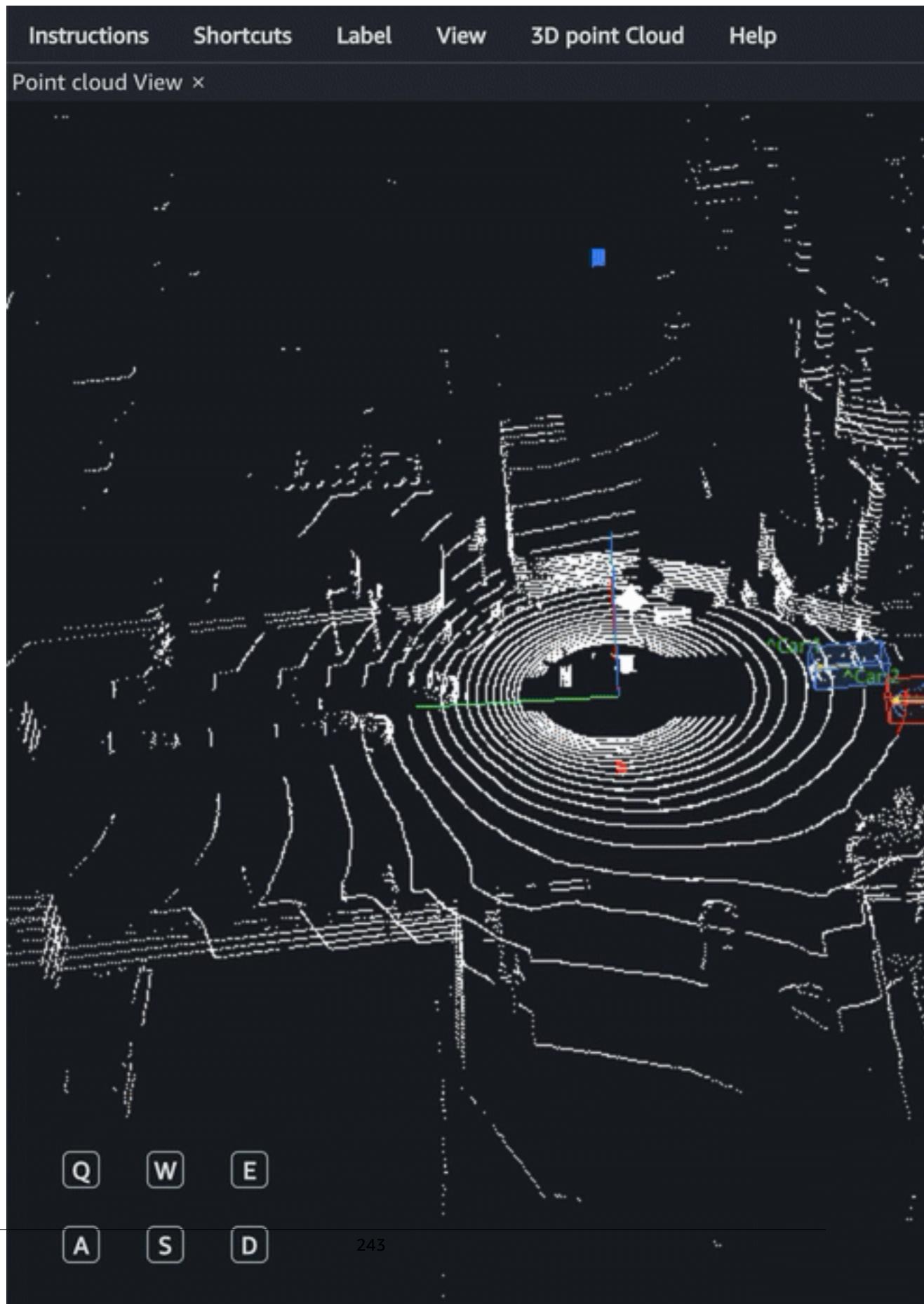
- [View the Worker Task Interface \(p. 241\)](#)
- [Create a 3D Point Cloud Object Tracking Labeling Job \(p. 249\)](#)
- [Create a 3D Point Cloud Object Tracking Adjustment or Verification Labeling Job \(p. 250\)](#)
- [Output Data Format \(p. 250\)](#)

### View the Worker Task Interface

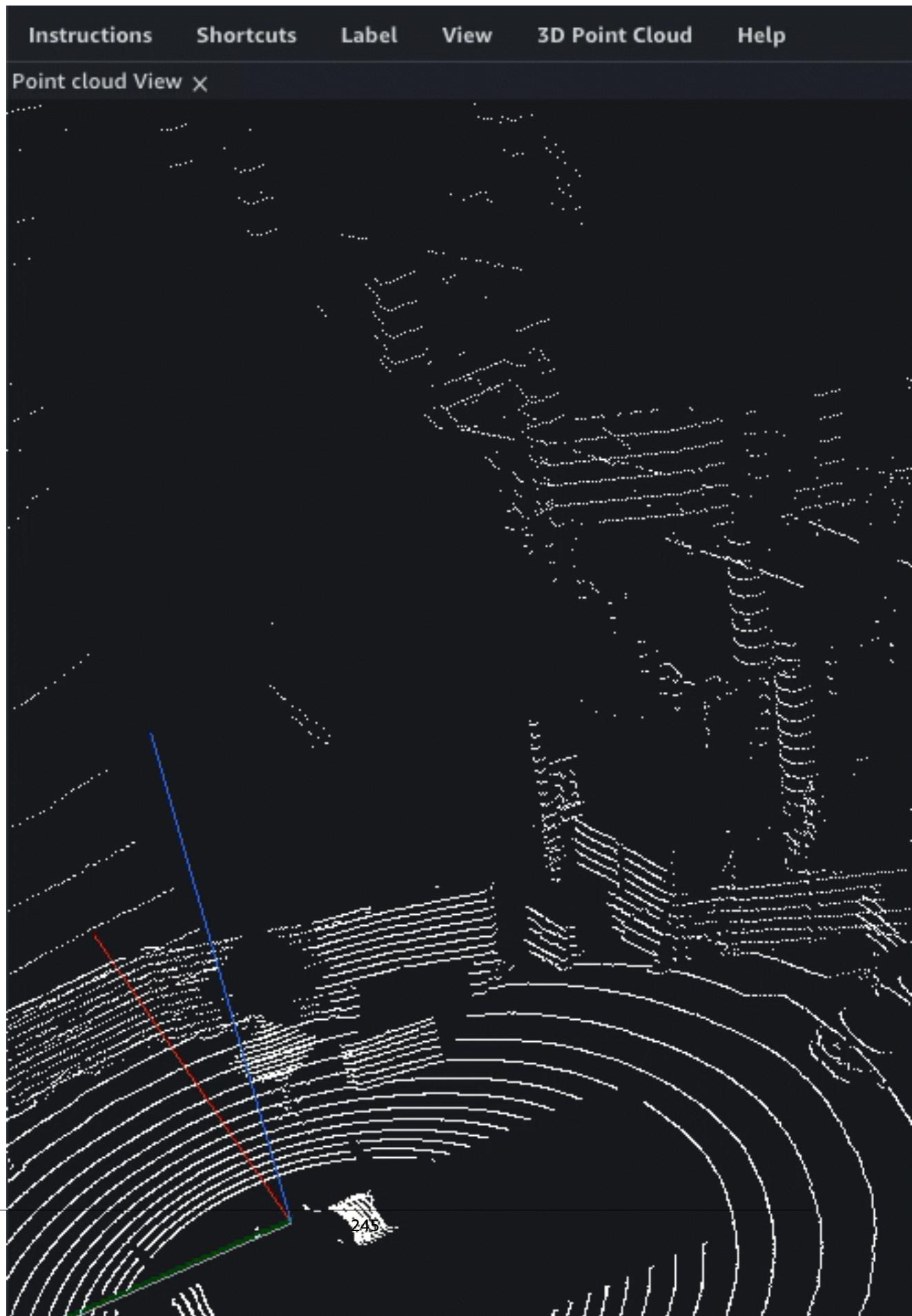
Ground Truth provides workers with a web portal and tools to complete your 3D point cloud object tracking annotation tasks. When you create the labeling job, you provide the Amazon Resource Name

(ARN) for a pre-built Ground Truth UI in the `HumanTaskUiArn` parameter. When you create a labeling job using this task type in the console, this UI is automatically used. You can preview and interact with the worker UI when you create a labeling job in the console. If you are a new user, it is recommended that you create a labeling job using the console to ensure your label attributes, point cloud frames, and if applicable, images, appear as expected.

The following is a GIF of the 3D point cloud object tracking worker task interface and demonstrates how the worker can navigate the point cloud frames in the sequence.



Once workers add a single cuboid, that cuboid is replicated in all frames of the sequence with the same ID. Once workers adjust the cuboid in another frame, Ground Truth will interpolate the movement of that object and adjust all cuboids between the manually adjusted frames. The following GIF demonstrates this interpolation feature. In the navigation bar on the bottom-left, red-areas indicate manually adjusted frames.



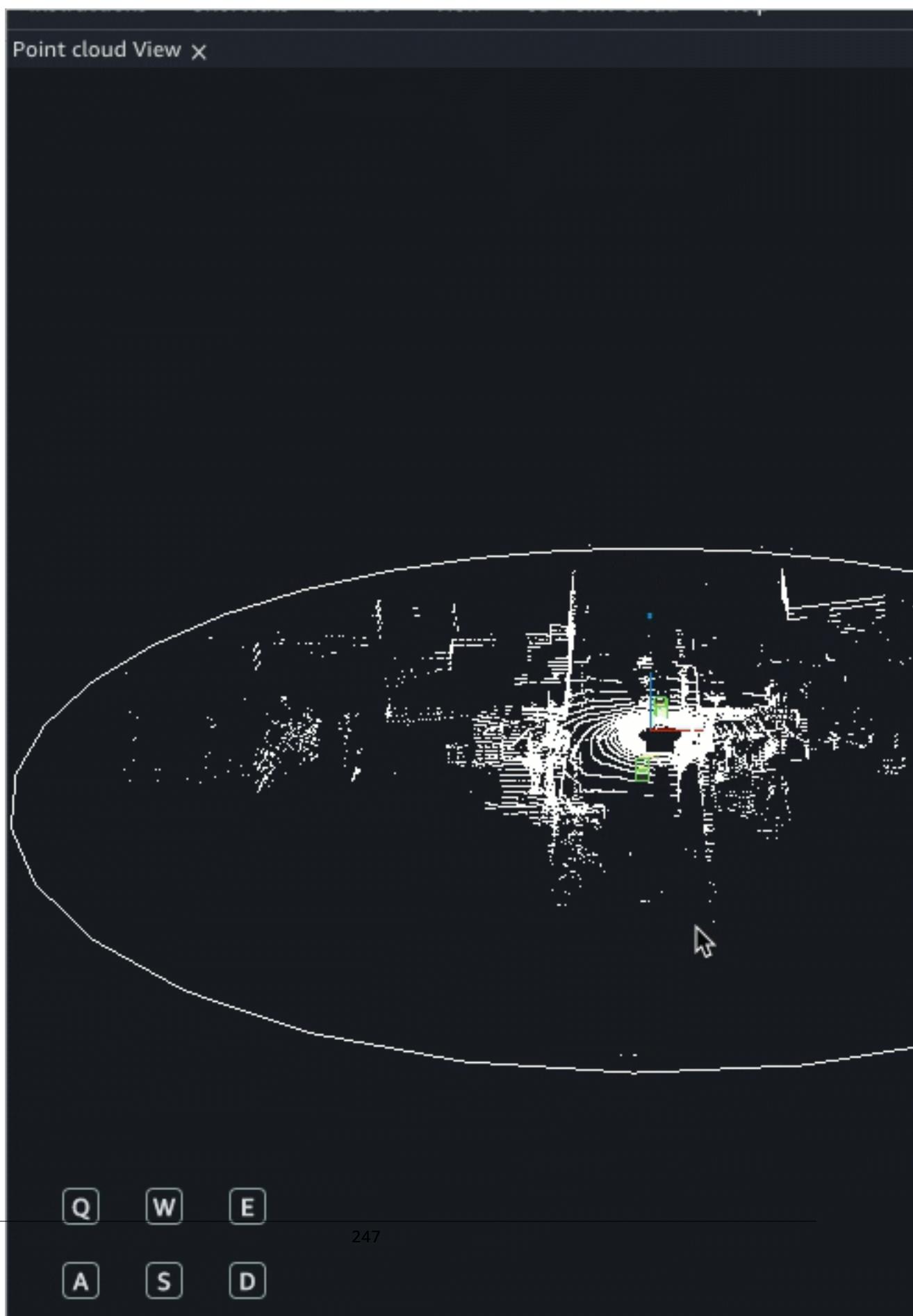
If you provide camera data for sensor fusion, images are matched up with scenes in point cloud frames. These images appear in the worker portal as shown in the following GIF.

Worker can navigate in the 3D scene using their keyboard and mouse. They can:

- Double click on specific objects in the point cloud to zoom into them.
- Use a mouse-scroller or trackpad to zoom in and out of the point cloud.
- Use both keyboard arrow keys and Q, E, A, and D keys to move Up, Down, Left, Right. Use keyboard keys W and S to zoom in and out.

Once a worker places a cuboids in the 3D scene, a side-view will appear with the three projected side views: top, side, and back. These side-views show points in and around the placed cuboid and help workers refine cuboid boundaries in that area. Workers can zoom in and out of each of those side-views using their mouse.

The following video demonstrates movements around the 3D point cloud and in the side-view.



Additional view options and features are available. See the [worker instruction page](#) for a comprehensive overview of the Worker UI.

### Worker Tools

Workers can navigate through the 3D point cloud by zooming in and out, and moving in all directions around the cloud using the mouse and keyboard shortcuts. If workers click on a point in the point cloud, the UI will automatically zoom into that area. Workers can use various tools to draw 3D cuboid around objects. For more information, see [Assistive Labeling Tools](#).

After workers have placed a 3D cuboid in the point cloud, they can adjust these cuboids to fit tightly around cars using a variety of views: directly in the 3D cuboid, in a side-view featuring three zoomed-in perspectives of the point cloud around the box, and if you include images for sensor fusion, directly in the 2D image.

View options that enable workers to easily hide or view label text, a ground mesh, and additional point attributes. Workers can also choose between perspective and orthogonal projections.

### Assistive Labeling Tools

Ground Truth helps workers annotate 3D point clouds faster and more accurately using UX, machine learning and computer vision powered assistive labeling tools for 3D point cloud object tracking tasks. The following assistive labeling tools are available for this task type:

- **Label autofill** – When a worker adds a cuboid to a frame, a cuboid with the same dimensions and orientation is automatically added to all frames in the sequence.
- **Label interpolation** – After a worker has labeled a single object in two frames, Ground Truth uses those annotations to interpolate the movement of that object between those two frames.
- **Bulk label and attribute management** – Workers can add, delete, and rename annotations, label category attributes, and frame attributes in bulk.
  - Workers can manually delete annotations for a given object before or after a frame. For example, a worker can delete all labels for an object after frame 10 if that object is no longer located in the scene after that frame.
  - If a worker accidentally bulk deletes all annotations for an object, they can add them back. For example, if a worker deletes all annotations for an object before frame 100, they can bulk add them to those frames.
  - Workers can rename a label in one frame and all 3D cuboids assigned that label are updated with the new name across all frames.
  - Workers can use bulk editing to add or edit label category attributes and frame attributes in multiple frames.
- **Snapping** – Workers can add a cuboid around an object and use a keyboard shortcut or menu option to have Ground Truth's autofit tool snap the cuboid tightly around the object's boundaries.
- **Fit to ground** – After a worker adds a cuboid to the 3D scene, the worker can automatically snap the cuboid to the ground. For example, the worker can use this feature to snap a cuboid to the road or sidewalk in the scene.
- **Multi-view labeling** – After a worker adds a 3D cuboid to the 3D scene, a side -panel displays front and two side perspectives to help the worker adjust the cuboid tightly around the object. Workers can annotation the 3D point cloud, the side panel and the adjustments appear in the other views in real time.
- **Sensor fusion** – If you provide data for sensor fusion, workers can adjust annotations in the 3D scenes and in 2D images, and the annotations will be projected into the other view in real time.
- **Auto-merge cuboids** – Workers can automatically merge two cuboids across all frames if they determine that cuboids with different labels actually represent a single object.
- **View options** – Enables workers to easily hide or view label text, a ground mesh, and additional point attributes like color or intensity. Workers can also choose between perspective and orthogonal projections.

## Create a 3D Point Cloud Object Tracking Labeling Job

You can create a 3D point cloud labeling job using the SageMaker console or API operation, [CreateLabelingJob](#). To create a labeling job for this task type you need the following:

- A sequence input manifest file. To learn how to create this type of manifest file, see [Create a Point Cloud Sequence Input Manifest \(p. 383\)](#). If you are a new user of Ground Truth 3D point cloud labeling modalities, we recommend that you review [Accepted Raw 3D Data Formats \(p. 375\)](#).
- A work team from a private or vendor workforce. You cannot use Amazon Mechanical Turk for 3D point cloud labeling jobs. To learn how to create workforces and work teams, see [Create and Manage Workforces \(p. 456\)](#).

Additionally, make sure that you have reviewed and satisfied the [Assign IAM Permissions to Use Ground Truth \(p. 440\)](#).

To learn how to create a labeling job using the console or an API, see the following sections.

### Create a Labeling Job (API)

This section covers details you need to know when you create a labeling job using the SageMaker API operation `CreateLabelingJob`. This API defines this operation for all Amazon SDKs. To see a list of language-specific SDKs supported for this operation, review the **See Also** section of [CreateLabelingJob](#).

[Create a Labeling Job \(API\) \(p. 338\)](#) provides an overview of the `CreateLabelingJob` operation. Follow these instructions and do the following while you configure your request:

- You must enter an ARN for `HumanTaskUiArn`. Use `arn:aws:sagemaker:<region>:394669845002:human-task-ui/PointCloudObjectTracking`. Replace `<region>` with the Amazon Region you are creating the labeling job in.

There should not be an entry for the `UiTemplatesS3Uri` parameter.

- Your `LabelAttributeName` must end in `-ref`. For example, `ot-labels-ref`.
- Your input manifest file must be a point cloud frame sequence manifest file. For more information, see [Create a Point Cloud Sequence Input Manifest \(p. 383\)](#).
- You specify your labels, label category and frame attributes, and worker instructions in a label category configuration file. For more information, see [Create a Labeling Category Configuration File with Label Category and Frame Attributes \(p. 348\)](#) to learn how to create this file.
- You need to provide pre-defined ARNs for the pre-annotation and post-annotation (ACS) Lambda functions. These ARNs are specific to the Amazon Region you use to create your labeling job.
  - To find the pre-annotation Lambda ARN, refer to `PreHumanTaskLambdaArn`. Use the Region you are creating your labeling job in to find the correct ARN that ends with `PRE-3DPointCloudObjectTracking`.
  - To find the post-annotation Lambda ARN, refer to `AnnotationConsolidationLambdaArn`. Use the Region you are creating your labeling job in to find the correct ARN that ends with `ACS-3DPointCloudObjectTracking`.
- The number of workers specified in `NumberOfHumanWorkersPerDataObject` should be 1.
- Automated data labeling is not supported for 3D point cloud labeling jobs. You should not specify values for parameters in `LabelingJobAlgorithmsConfig`.
- 3D point cloud object tracking labeling jobs can take multiple hours to complete. You can specify a longer time limit for these labeling jobs in `TaskTimeLimitInSeconds` (up to 7 days, or 604,800 seconds).

## Create a Labeling Job (Console)

You can follow the instructions [Create a Labeling Job \(Console\) \(p. 336\)](#) in order to learn how to create a 3D point cloud object tracking labeling job in the SageMaker console. While you are creating your labeling job, be aware of the following:

- Your input manifest file must be a sequence manifest file. For more information, see [Create a Point Cloud Sequence Input Manifest \(p. 383\)](#).
- Optionally, you can provide label category attributes. Workers can assign one or more of these attributes to annotations to provide more information about that object. For example, you might want to use the attribute *occluded* to have workers identify when an object is partially obstructed.
- Automated data labeling and annotation consolidation are not supported for 3D point cloud labeling tasks.
- 3D point cloud object tracking labeling jobs can take multiple hours to complete. You can specify a longer time limit for these labeling jobs when you select your work team (up to 7 days, or 604800 seconds).

## Create a 3D Point Cloud Object Tracking Adjustment or Verification Labeling Job

You can create an adjustment and verification labeling job using the Ground Truth console or `CreateLabelingJob` API. To learn more about adjustment and verification labeling jobs, and to learn how to create one, see [Verify and Adjust Labels \(p. 293\)](#).

When you create an adjustment labeling job, your input data to the labeling job can include labels, and yaw, pitch, and roll measurements from a previous labeling job or external source. In the adjustment job, pitch, and roll will be visualized in the worker UI, but cannot be modified. Yaw is adjustable.

Ground Truth uses Tait-Bryan angles with the following intrinsic rotations to visualize yaw, pitch and roll in the worker UI. First, rotation is applied to the vehicle according to the z-axis (yaw). Next, the rotated vehicle is rotated according to the intrinsic y'-axis (pitch). Finally, the vehicle is rotated according to the intrinsic x''-axis (roll).

## Output Data Format

When you create a 3D point cloud object tracking labeling job, tasks are sent to workers. When these workers complete their tasks, their annotations are written to the Amazon S3 bucket you specified when you created the labeling job. The output data format determines what you see in your Amazon S3 bucket when your labeling job status (`LabelingJobStatus`) is Completed.

If you are a new user of Ground Truth, see [Output Data \(p. 404\)](#) to learn more about the Ground Truth output data format. To learn about the 3D point cloud object tracking output data format, see [3D Point Cloud Object Tracking Output \(p. 424\)](#).

## 3D Point Cloud Semantic Segmentation

Semantic segmentation involves classifying individual points of a 3D point cloud into pre-specified categories. Use this task type when you want workers to create a point-level semantic segmentation mask for 3D point clouds. For example, if you specify the classes *car*, *pedestrian*, and *bike*, workers select one class at a time, and color all of the points that this class applies to the same color in the point cloud.

For this task type, the data object that workers label is a single point cloud frame. Ground Truth generates a 3D point cloud visualization using point cloud data you provide. You can also provide camera data to give workers more visual information about scenes in the frame, and to help workers paint objects. When a worker paints an object in either the 2D image or the 3D point cloud, the paint shows up in the other view.

You can adjust annotations created in a 3D point cloud object detection labeling job using the 3D point cloud semantic segmentation adjustment task type.

If you are a new user of the Ground Truth 3D point cloud labeling modality, we recommend you review [3D Point Cloud Labeling Jobs Overview \(p. 259\)](#). This labeling modality is different from other Ground Truth task types, and this topic provides an overview of important details you should be aware of when creating a 3D point cloud labeling job.

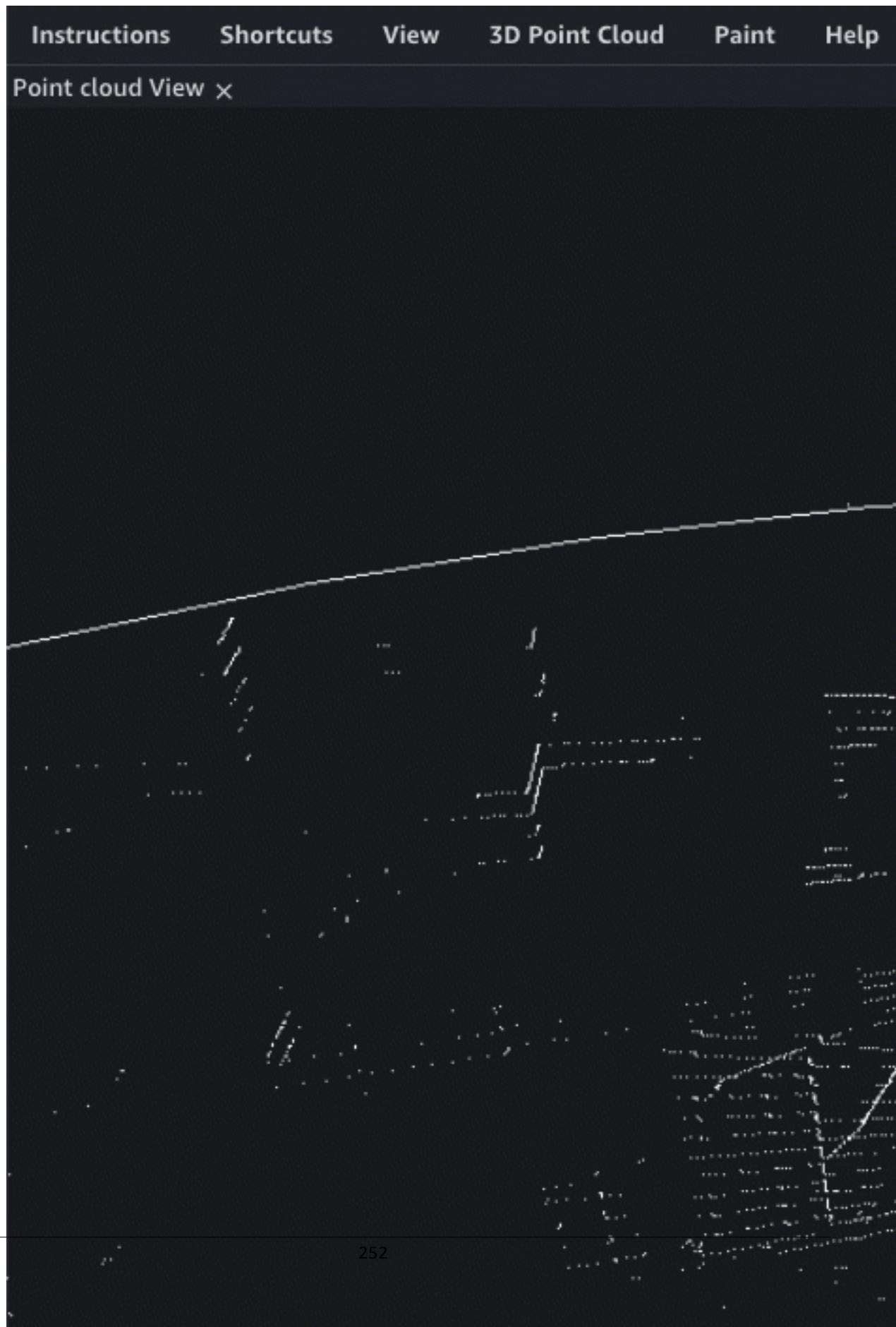
### Topics

- [View the Worker Task Interface \(p. 251\)](#)
- [Create a 3D Point Cloud Semantic Segmentation Labeling Job \(p. 257\)](#)
- [Create a 3D Point Cloud Semantic Segmentation Adjustment or Verification Labeling Job \(p. 258\)](#)
- [Output Data Format \(p. 258\)](#)

### [View the Worker Task Interface](#)

Ground Truth provides workers with a web portal and tools to complete your 3D point cloud semantic segmentation annotation tasks. When you create the labeling job, you provide the Amazon Resource Name (ARN) for a pre-built Ground Truth UI in the `HumanTaskUiArn` parameter. When you create a labeling job using this task type in the console, this UI is automatically used. You can preview and interact with the worker UI when you create a labeling job in the console. If you are a new user, it is recommended that you create a labeling job using the console to ensure your label attributes, point cloud frames, and if applicable, images, appear as expected.

The following is a GIF of the 3D point cloud semantic segmentation worker task interface. If you provide camera data for sensor fusion, images are matched with scenes in the point cloud frame. Workers can paint objects in either the 3D point cloud or the 2D image, and the paint appears in the corresponding location in the other medium. These images appear in the worker portal as shown in the following GIF.



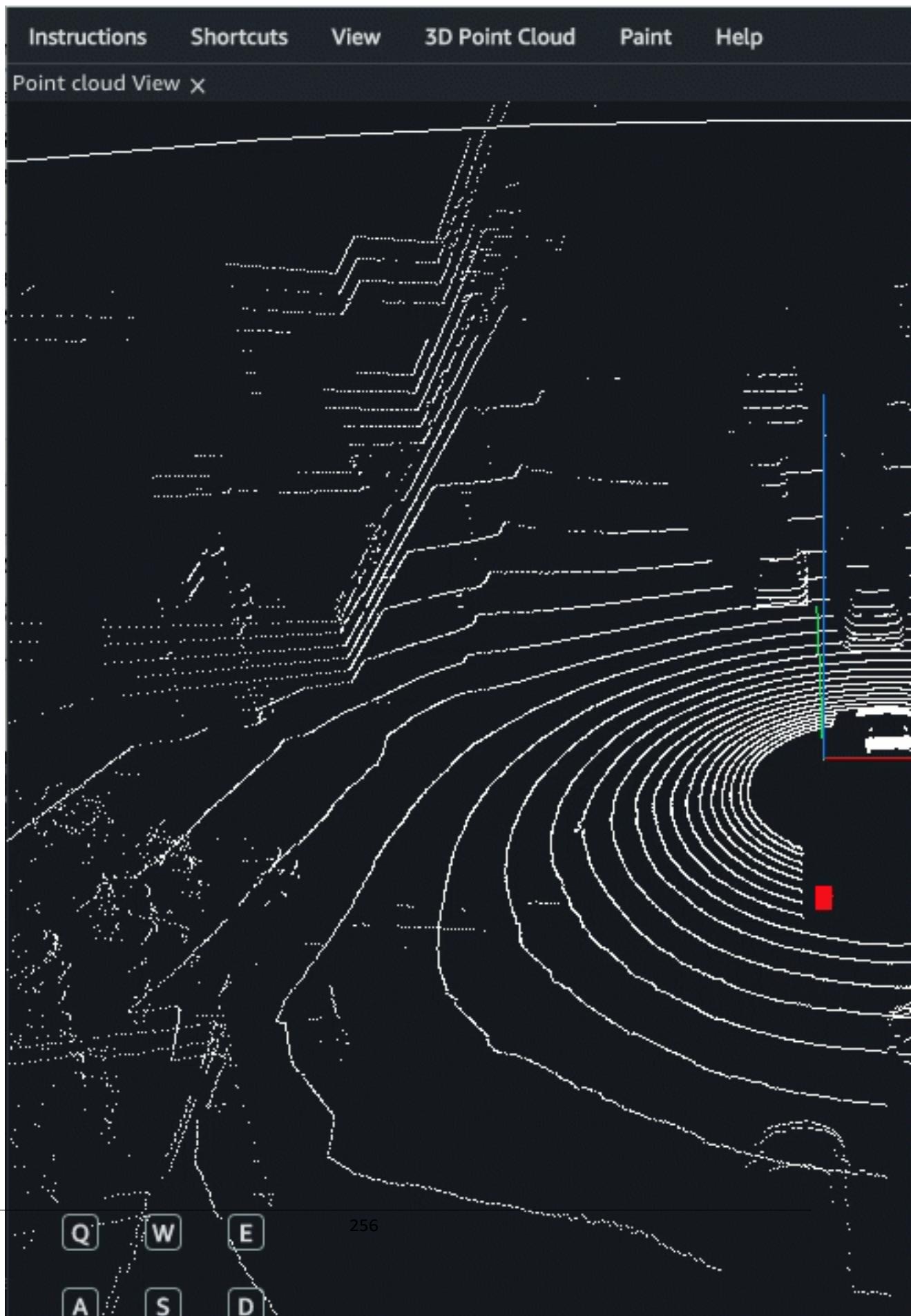
Worker can navigate in the 3D scene using their keyboard and mouse. They can:

- Double click on specific objects in the point cloud to zoom into them.
- Use a mouse-scroller or trackpad to zoom in and out of the point cloud.
- Use both keyboard arrow keys and Q, E, A, and D keys to move Up, Down, Left, Right. Use keyboard keys W and S to zoom in and out.

The following video demonstrates movements around the 3D point cloud. Workers can hide and re-expand all side views and menus. In this GIF, the side-views and menus have been collapsed.



The following GIF demonstrates how a worker can label multiple objects quickly, refine painted objects using the Unpaint option and then view only points that have been painted.



Additional view options and features are available. See the [worker instruction page](#) for a comprehensive overview of the Worker UI.

### Worker Tools

Workers can navigate through the 3D point cloud by zooming in and out, and moving in all directions around the cloud using the mouse and keyboard shortcuts. When you create a semantic segmentation job, workers have the following tools available to them:

- A paint brush to paint and unpaint objects. Workers paint objects by selecting a label category and then painting in the 3D point cloud. Workers unpaint objects by selecting the Unpaint option from the label category menu and using the paint brush to erase paint.
- A polygon tool that workers can use to select and paint an area in the point cloud.
- A background paint tool, which enables workers to paint behind objects they have already annotated without altering the original annotations. For example, workers might use this tool to paint the road after painting all of the cars on the road.
- View options that enable workers to easily hide or view label text, a ground mesh, and additional point attributes like color or intensity. Workers can also choose between perspective and orthogonal projections.

## Create a 3D Point Cloud Semantic Segmentation Labeling Job

You can create a 3D point cloud labeling job using the SageMaker console or API operation, [CreateLabelingJob](#). To create a labeling job for this task type you need the following:

- A single-frame input manifest file. To learn how to create this type of manifest file, see [Create a Point Cloud Frame Input Manifest File \(p. 376\)](#). If you are a new user of Ground Truth 3D point cloud labeling modalities, we recommend that you review [Accepted Raw 3D Data Formats \(p. 375\)](#).
- A work team from a private or vendor workforce. You cannot use Amazon Mechanical Turk workers for 3D point cloud labeling jobs. To learn how to create workforces and work teams, see [Create and Manage Workforces \(p. 456\)](#).
- A label category configuration file. For more information, see [Create a Labeling Category Configuration File with Label Category and Frame Attributes \(p. 348\)](#).

Additionally, make sure that you have reviewed and satisfied the [Assign IAM Permissions to Use Ground Truth \(p. 440\)](#).

Use one of the following sections to learn how to create a labeling job using the console or an API.

### Create a Labeling Job (Console)

You can follow the instructions [Create a Labeling Job \(Console\) \(p. 336\)](#) in order to learn how to create a 3D point cloud semantic segmentation labeling job in the SageMaker console. While you are creating your labeling job, be aware of the following:

- Your input manifest file must be a single-frame manifest file. For more information, see [Create a Point Cloud Frame Input Manifest File \(p. 376\)](#).
- Automated data labeling and annotation consolidation are not supported for 3D point cloud labeling tasks.
- 3D point cloud semantic segmentation labeling jobs can take multiple hours to complete. You can specify a longer time limit for these labeling jobs when you select your work team (up to 7 days, or 604800 seconds).

## Create a Labeling Job (API)

This section covers details you need to know when you create a labeling job using the SageMaker API operation `CreateLabelingJob`. This API defines this operation for all Amazon SDKs. To see a list of language-specific SDKs supported for this operation, review the **See Also** section of `CreateLabelingJob`.

The page, [Create a Labeling Job \(API\) \(p. 338\)](#), provides an overview of the `CreateLabelingJob` operation. Follow these instructions and do the following while you configure your request:

- You must enter an ARN for `HumanTaskUiArn`. Use `arn:aws:sagemaker:<region>:394669845002:human-task-ui/PointCloudSemanticSegmentation`. Replace `<region>` with the Amazon Region you are creating the labeling job in.

There should not be an entry for the `UiTemplateS3Uri` parameter.

- Your `LabelAttributeName` must end in `-ref`. For example, `ss-labels-ref`.
- Your input manifest file must be a single-frame manifest file. For more information, see [Create a Point Cloud Frame Input Manifest File \(p. 376\)](#).
- You specify your labels and worker instructions in a label category configuration file. See [Create a Labeling Category Configuration File with Label Category and Frame Attributes \(p. 348\)](#) to learn how to create this file.
- You need to provide a pre-defined ARNs for the pre-annotation and post-annotation (ACS) Lambda functions. These ARNs are specific to the Amazon Region you use to create your labeling job.
  - To find the pre-annotation Lambda ARN, refer to `PreHumanTaskLambdaArn`. Use the Region you are creating your labeling job in to find the correct ARN. For example, if you are creating your labeling job in us-east-1, the ARN will be `arn:aws:lambda:us-east-1:432418664414:function:PRE-3DPointCloudSemanticSegmentation`.
  - To find the post-annotation Lambda ARN, refer to `AnnotationConsolidationLambdaArn`. Use the Region you are creating your labeling job in to find the correct ARN. For example, if you are creating your labeling job in us-east-1, the ARN will be `arn:aws:lambda:us-east-1:432418664414:function:ACS-3DPointCloudSemanticSegmentation`.
- The number of workers specified in `NumberOfHumanWorkersPerDataObject` should be 1.
- Automated data labeling is not supported for 3D point cloud labeling jobs. You should not specify values for parameters in `LabelingJobAlgorithmsConfig`.
- 3D point cloud semantic segmentation labeling jobs can take multiple hours to complete. You can specify a longer time limit for these labeling jobs in `TaskTimeLimitInSeconds` (up to 7 days, or 604800 seconds).

## Create a 3D Point Cloud Semantic Segmentation Adjustment or Verification Labeling Job

You can create an adjustment and verification labeling job using the Ground Truth console or `CreateLabelingJob` API. To learn more about adjustment and verification labeling jobs, and to learn how to create one, see [Verify and Adjust Labels \(p. 293\)](#).

### Output Data Format

When you create a 3D point cloud semantic segmentation labeling job, tasks are sent to workers. When these workers complete their tasks, their annotations are written to the Amazon S3 bucket you specified when you created the labeling job. The output data format determines what you see in your Amazon S3 bucket when your labeling job status (`LabelingJobStatus`) is `Completed`.

If you are a new user of Ground Truth, see [Output Data \(p. 404\)](#) to learn more about the Ground Truth output data format. To learn about the 3D point cloud object detection output data format, see [3D Point Cloud Semantic Segmentation Output \(p. 420\)](#).

## 3D Point Cloud Labeling Jobs Overview

This topic provides an overview of the unique features of a Ground Truth 3D point cloud labeling job. You can use the 3D point cloud labeling jobs to have workers label objects in a 3D point cloud generated from a 3D sensors like LiDAR and depth cameras or generated from 3D reconstruction by stitching images captured by an agent like a drone.

### Job Pre-processing Time

When you create a 3D point cloud labeling job, you need to provide an [input manifest file \(p. 374\)](#). The input manifest file can be:

- A *frame input manifest file* that has a single point cloud frame on each line.
- A *sequence input manifest file* that has a single sequence on each line. A sequence is defined as a temporal series of point cloud frames.

For both types of manifest files, *job pre-processing time* (that is, the time before Ground Truth starts sending tasks to your workers) depends on the total number and size of point cloud frames you provide in your input manifest file. For frame input manifest files, this is the number of lines in your manifest file. For sequence manifest files, this is the number of frames in each sequence multiplied by the total number of sequences, or lines, in your manifest file.

Additionally, the number of points per point cloud and the number of fused sensor data objects (like images) factor into job pre-processing times. On average, Ground Truth can pre-process 200 point cloud frames in approximately 5 minutes. If you create a 3D point cloud labeling job with a large number of point cloud frames, you might experience longer job pre-processing times. For example, if you create a sequence input manifest file with 4 point cloud sequences, and each sequence contains 200 point clouds, Ground Truth pre-processes 800 point clouds and so your job pre-processing time might be around 20 minutes. During this time, your labeling job status is `InProgress`.

While your 3D point cloud labeling job is pre-processing, you receive CloudWatch messages notifying you of the status of your job. To identify these messages, search for `3D_POINT_CLOUD_PROCESSING_STATUS` in your labeling job logs.

For **frame input manifest files**, your CloudWatch logs will have a message similar to the following:

```
{  
    "labeling-job-name": "example-point-cloud-labeling-job",  
    "event-name": "3D_POINT_CLOUD_PROCESSING_STATUS",  
    "event-log-message": "datasetObjectId from: 0 to 10, status: IN_PROGRESS"  
}
```

The event log message, `datasetObjectId from: 0 to 10, status: IN_PROGRESS` identifies the number of frames from your input manifest that have been processed. You receive a new message every time a frame has been processed. For example, after a single frame has processed, you receive another message that says `datasetObjectId from: 1 to 10, status: IN_PROGRESS`.

For **sequence input manifest files**, your CloudWatch logs will have a message similar to the following:

```
{  
    "labeling-job-name": "example-point-cloud-labeling-job",  
    "event-name": "3D_POINT_CLOUD_PROCESSING_STATUS",  
    "event-log-message": "datasetObjectId: 0, status: IN_PROGRESS"  
}
```

The event log message, `datasetObjectId from: 0, status: IN_PROGRESS` identifies the number of sequences from your input manifest that have been processed. You receive a new message every

time a sequence has been processed. For example, after a single sequence has processed, you receive a message that says `datasetObjectId from: 1, status: IN_PROGRESS` as the next sequence begins processing.

## Job Completion Times

3D point cloud labeling jobs can take workers hours to complete. You can set the total amount of time that workers can work on each task when you create a labeling job. The maximum time you can set for workers to work on tasks is 7 days. The default value is 3 days.

It is strongly recommended that you create tasks that workers can complete within 12 hours. Workers must keep the worker UI open while working on a task. They can save work as they go and Ground Truth will save their work every 15 minutes.

When using the SageMaker `CreateLabelingJob` API operation, set the total time a task is available to workers in the `TaskTimeLimitInSeconds` parameter of `HumanTaskConfig`.

When you create a labeling job in the console, you can specify this time limit when you select your workforce type and your work team.

## Workforces

When you create a 3D point cloud labeling job, you need to specify a work team that will complete your point cloud annotation tasks. You can choose a work team from a private workforce of your own workers, or from a vendor workforce that you select in the Amazon Web Services Marketplace. You cannot use the Amazon Mechanical Turk workforce for 3D point cloud labeling jobs.

To learn more about vendor workforce, see [Managing Vendor Workforces \(p. 460\)](#).

To learn how to create and manage a private workforce, see [Use a Private Workforce \(p. 461\)](#).

## Worker User Interface (UI)

Ground Truth provides a worker user interface (UI), tools, and assistive labeling features to help workers complete your 3D point cloud labeling tasks.

You can preview the worker UI when you create a labeling job in the console.

When you create a labeling job using the API operation `CreateLabelingJob`, you must provide an ARN provided by Ground Truth in the parameter `HumanTaskUiArn` to specify the worker UI for your task type. You can use `HumanTaskUiArn` with the SageMaker `RenderUiTemplate` API operation to preview the worker UI.

You provide worker instructions, labels, and optionally, label category attributes that are displayed in the worker UI.

## Label Category Attributes

When you create a 3D point cloud object tracking or object detection labeling job, you can add one or more *label category attributes*. You can add *frame attributes* to all 3D point cloud task types:

- **Label category attribute** – A list of options (strings), a free form text box, or a numeric field associated with one or more labels. It is used by workers to provide metadata about a label.
- **Frame attribute** – A list of options (strings), a free form text box, or a numeric field that appears on each point cloud frame a worker is sent to annotate. It is used by workers to provide metadata about frames.

Additionally, you can use label and frame attributes to have workers verify labels in a 3D point cloud label verification job.

Use the following sections to learn more about these attributes. To learn how to add label category and frame attributes to a labeling job, use the **Create Labeling Job** section on the [task type page](#) of your choice.

### [Label Category Attributes](#)

Add label category attributes to labels to give workers the ability to provide more information about the annotations they create. A label category attribute is added to an individual label, or to all labels. When a label category attribute is applied to all labels it is referred to as a *global label category attribute*.

For example, if you add the label category *car*, you might also want to capture additional data about your labeled cars, such as if they are occluded or the size of the car. You can capture this metadata using label category attributes. In this example, if you added the attribute *occluded* to the car label category, you can assign *partial*, *completely*, *no* to the *occluded* attribute and enable workers to select one of these options.

When you create a label verification job, you add labels category attributes to each label you want workers to verify.

### [Frame Attributes](#)

Add frame attributes to give workers the ability to provide more information about individual point cloud frames. You can specify up to 10 frame attributes, and these attributes will appear on all frames.

For example, you can add a frame attribute that allows workers to enter a number. You may want to use this attribute to have workers identify the number of objects they see in a particular frame.

In another example, you may want to provide a free-form text box to give workers the ability to provide a free form answer to a question.

When you create a label verification job, you can add one or more frame attributes to ask workers to provide feedback on all labels in a point cloud frame.

### [Worker Instructions](#)

You can provide worker instructions to help your workers complete your point cloud labeling tasks. You might want to use these instructions to do the following:

- Best practices and things to avoid when annotating objects.
- Explanation of the label category attributes provided (for object detection and object tracking tasks), and how to use them.
- Advice on how to save time while labeling by using keyboard shortcuts.

You can add your worker instructions using the SageMaker console while creating a labeling job. If you create a labeling job using the API operation `CreateLabelingJob`, you specify worker instructions in your label category configuration file.

In addition to your instructions, Ground Truth provides a link to help workers navigate and use the worker portal. View these instructions by selecting the task type on [Worker Instructions \(p. 262\)](#).

### [Declining Tasks](#)

Workers are able to decline tasks.

Workers decline a task if the instructions are not clear, input data is not displaying correctly, or if they encounter some other issue with the task. If the number of workers per dataset object (`NumberOfHumanWorkersPerDataObject`) decline the task, the data object is marked as expired and will not be sent to additional workers.

## 3D Point Cloud Labeling Job Permission Requirements

When you create a 3D point cloud labeling job, in addition to the permission requirements found in [Assign IAM Permissions to Use Ground Truth \(p. 440\)](#), you must add a CORS policy to your S3 bucket that contains your input manifest file.

### Add a CORS Permission Policy to S3 Bucket

When you create a 3D point cloud labeling job, you specify buckets in S3 where your input data and manifest file are located and where your output data will be stored. These buckets may be the same. You must attach the following Cross-origin resource sharing (CORS) policy to your input and output buckets. If you use the Amazon S3 console to add the policy to your bucket, you must use the JSON format.

#### JSON

```
[  
  {  
    "AllowedHeaders": [  
      "*"  
    ],  
    "AllowedMethods": [  
      "GET",  
      "HEAD",  
      "PUT"  
    ],  
    "AllowedOrigins": [  
      "*"  
    ],  
    "ExposeHeaders": [  
      "Access-Control-Allow-Origin"  
    ],  
    "MaxAgeSeconds": 3000  
  }  
]
```

#### XML

```
<?xml version="1.0" encoding="UTF-8"?>  
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">  
  <CORSRule>  
    <AllowedOrigin>*</AllowedOrigin>  
    <AllowedMethod>GET</AllowedMethod>  
    <AllowedMethod>HEAD</AllowedMethod>  
    <AllowedMethod>PUT</AllowedMethod>  
    <MaxAgeSeconds>3000</MaxAgeSeconds>  
    <ExposeHeader>Access-Control-Allow-Origin</ExposeHeader>  
    <AllowedHeader>*</AllowedHeader>  
  </CORSRule>  
</CORSConfiguration>
```

To learn how to add a CORS policy to an S3 bucket, see [How do I add cross-domain resource sharing with CORS?](#) in the Amazon Simple Storage Service Console User Guide.

## Worker Instructions

This topic provides an overview of the Ground Truth worker portal and the tools available to complete your 3D Point Cloud labeling task. First, select the type of task you are working on from **Topics**.

For adjustment jobs, select the original labeling job task type that produced the labels you are adjusting. Review and adjust the labels in your task as needed.

**Important**

It is recommended that you complete your task using a Google Chrome or Firefox web browser.

**Topics**

- [3D Point Cloud Semantic Segmentation \(p. 263\)](#)
- [3D Point Cloud Object Detection \(p. 272\)](#)
- [3D Point Cloud Object Tracking \(p. 282\)](#)

## 3D Point Cloud Semantic Segmentation

Use this page to become familiarize with the user interface and tools available to complete your 3D point cloud semantic segmentation task.

**Topics**

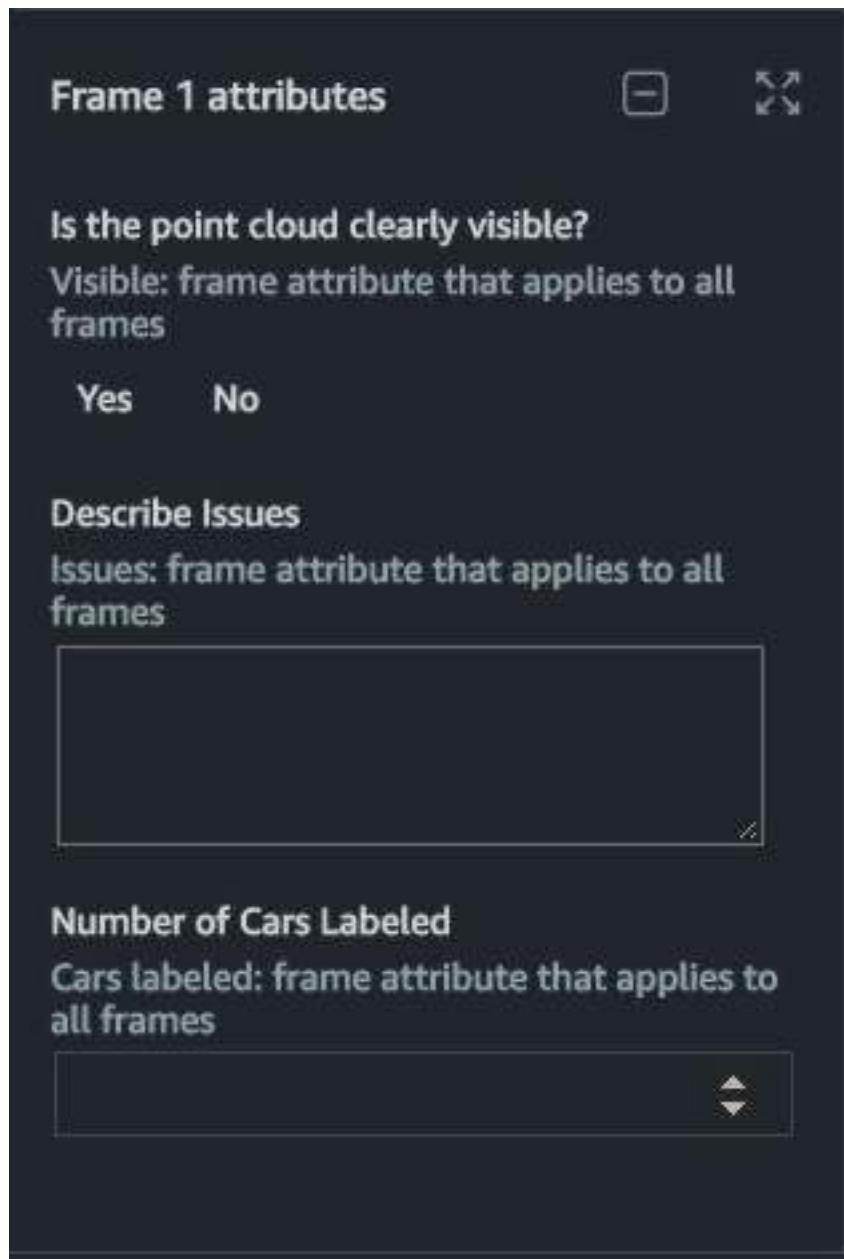
- [Your Task \(p. 263\)](#)
- [Navigate the UI \(p. 268\)](#)
- [Icon Guide \(p. 270\)](#)
- [Shortcuts \(p. 271\)](#)
- [Release, Stop and Resume, and Decline Tasks \(p. 271\)](#)
- [Saving Your Work and Submitting \(p. 272\)](#)

### Your Task

When you work on a 3D point cloud semantic segmentation task, you need to select a category from the **Annotations** menu on the right side of your worker portal using the drop down menu **Label Categories**. After you've selected a category, use the paint brush and polygon tools to paint each object in the 3D point cloud that this category applies to. For example, if you select the category **Car**, you would use these tools to paint all of the cars in the point cloud. The following video demonstrates how to use the paint brush tool to paint an object.

If you see one or more images in your worker portal, you can paint in the images or paint in the 3D point cloud and the paint will show up in the other medium.

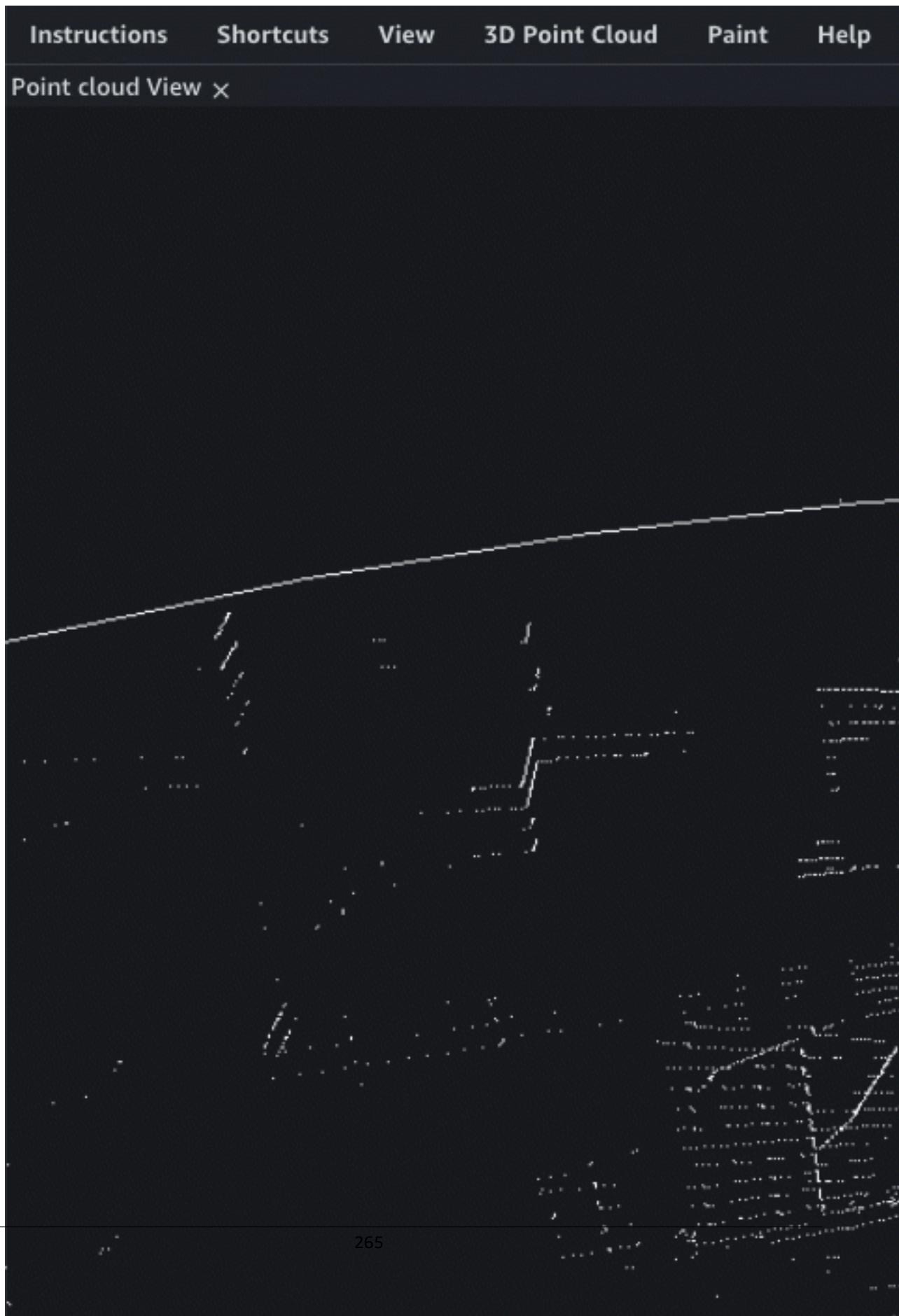
You may see frame attributes under the **Labels** menu. Use these attribute prompts to enter additional information about the point cloud.



**Important**

If you see that objects have already been painted when you open the task, adjust those annotations.

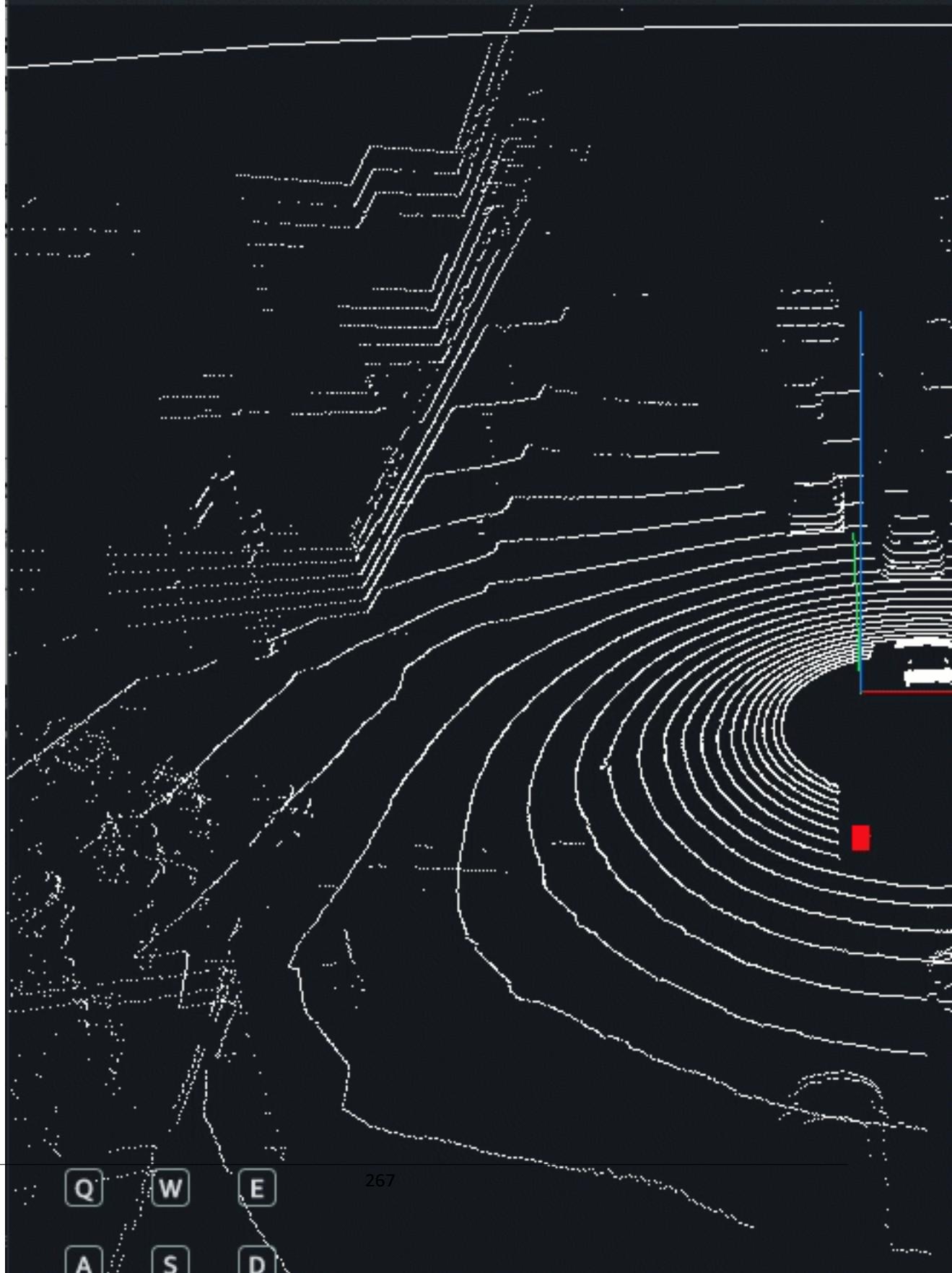
The following video includes an image that can be annotated. You may not see an image in your task.



After you've painted one or more objects using a label category, you can select that category from the Label Category menu on the right to only view points painted for that category.

Instructions   Shortcuts   View   **3D Point Cloud**   Paint   Help

Point cloud View X



## Navigate the UI

You can navigate in the 3D scene using their keyboard and mouse. You can:

- Double click on specific objects in the point cloud to zoom into them.
- Use a mouse-scroller or trackpad to zoom in and out of the point cloud.
- Use both keyboard arrow keys and Q, E, A, and D keys to move Up, Down, Left, Right. Use keyboard keys W and S to zoom in and out.

The following video demonstrates movements around the 3D point cloud and in the side-view. You can hide and re-expand all side views using the full screen icon. In this GIF, the side-views and menus have been collapsed.



When you are in the worker UI, you see the following menus:

- **Instructions** – Review these instructions before starting your task.
- **Shortcuts** – Use this menu to view keyboard shortcuts that you can use to navigate the point cloud and use the annotation tools provided.
- **View** – Use this menu to toggle different view options on and off. For example, you can use this menu to add a ground mesh to the point cloud, and to choose the projection of the point cloud.
- **3D Point Cloud** – Use this menu to add additional attributes to the points in the point cloud, such as color, and pixel intensity. Note that some or all of these options may not be available.
- **Paint** – Use this menu to modify the functionality of the paint brush.

When you open a task, the move scene icon is on, and you can move around the point cloud using your mouse and the navigation buttons in the point cloud area of the screen. To return to the original view you see when you first opened the task, choose the reset scene icon.

After you select the paint icon, you can add paint to the point cloud and images (if included). You must select the move scene icon again to move to another area in the 3D point cloud or image.

To collapse all panels on the right and make the 3D point cloud full screen, select the full screen icon.

For the camera images and side-panels, you have the following view options:

- **C** – View the camera angle on point cloud view.
- **F** – View the frustum, or field of view, of the camera used to capture that image on point cloud view.
- **P** – View the point cloud overlaid on the image.

## Icon Guide

Use this table to learn about the icons available in your worker task portal.

Icon	Name	Description
	brush	Choose this icon to turn on the brush tool. To use with this tool, choose and move over the objects that you want to paint with your mouse. After you choose it, everything you paint be associated with the category you chose.
	polygon	Choose this icon to use the polygon paint tool. Use this tool to draw polygons around objects that you want to paint. After you choose it, everything you draw a polygon around will be associated with the category you have chosen.
	reset scene	Choose this icon to reset the view of the point cloud, side panels, and if applicable, all images to their original position when the task was first opened.
	move scene	Choose this icon to move the scene. By default, this icon will be selected when you first start a task.

Icon	Name	Description
	full screen	Choose this icon to make the 3D point cloud visualization full screen, and to collapse all side panels.
	ruler	<p>Use this icon to measure distances, in meters, in the point cloud. You may want to use this tool if your instructions ask you to annotate all objects in a given distance from the center of the cuboid or the object used to capture data.</p> <p>When you select this icon, you can place the starting point (first marker) anywhere in the point cloud by selecting it with your mouse. The tool will automatically use interpolation to place a marker on the closest point within threshold distance to the location you select, otherwise the marker will be placed on ground. If you place a starting point by mistake, you can use the Escape key to revert marker placement.</p> <p>After you place the first marker, you see a dotted line and a dynamic label that indicates the distance you have moved away from the first marker. Click somewhere else on the point cloud to place a second marker. When you place the second marker, the dotted line becomes solid, and the distance is set.</p> <p>After you set a distance, you can edit it by selecting either marker. You can delete a ruler by selecting anywhere on the ruler and using the Delete key on your keyboard.</p>

## Shortcuts

The shortcuts listed in the **Shortcuts** menu can help you navigate the 3D point cloud and use the paint tool.

Before you start your task, it is recommended that you review the **Shortcuts** menu and become acquainted with these commands.

## Release, Stop and Resume, and Decline Tasks

When you open the labeling task, three buttons on the top right allow you to decline the task (**Decline task**), release it (**Release task**), and stop and resume it at a later time (**Stop and resume later**). The following list describes what happens when you select one of these options:

- **Decline task:** You should only decline a task if something is wrong with the task, such as an issue with the 3D point cloud, images or the UI. If you decline a task, you will not be able to return to the task.
- **Release Task:** If you release a task, you lose all work done on that task. When the task is released, other workers on your team can pick it up. If enough workers pick up the task, you may not be able to return to it. When you select this button and then select **Confirm**, you are returned to the worker portal. If the task is still available, its status will be **Available**. If other workers pick it up, it will disappear from your portal.
- **Stop and resume later:** You can use the **Stop and resume later** button to stop working and return to the task at a later time. You should use the **Save** button to save your work before you select **Stop and**

**resume later.** When you select this button and then select **Confirm**, you are returned to the worker portal, and the task status is **Stopped**. You can select the same task to resume work on it.

Be aware that the person that creates your labeling tasks specifies a time limit in which all tasks must be completed by. If you do not return to and complete this task within that time limit, it will expire and your work will not be submitted. Contact your administrator for more information.

## Saving Your Work and Submitting

You should periodically save your work. Ground Truth will automatically save your work every 15 minutes.

When you open a task, you must complete your work on it before pressing **Submit**.

## 3D Point Cloud Object Detection

Use this page to become familiar with the user interface and tools available to complete your 3D point cloud object detection task.

### Topics

- [Your Task \(p. 272\)](#)
- [Navigate the UI \(p. 274\)](#)
- [Icon Guide \(p. 280\)](#)
- [Shortcuts \(p. 281\)](#)
- [Release, Stop and Resume, and Decline Tasks \(p. 281\)](#)
- [Saving Your Work and Submitting \(p. 282\)](#)

### Your Task

When you work on a 3D point cloud object detection task, you need to select a category from the **Annotations** menu on the right side of your worker portal using the **Label Categories** menu. After you've chosen a category, use the add cuboid and fit cuboid tools to fit a cuboid around objects in the 3D point cloud that this category applies to. After you place a cuboid, you can modify its dimensions, location, and orientation directly in the point cloud, and the three panels shown on the right.

If you see one or more images in your worker portal, you can also modify cuboids in the images or in the 3D point cloud and the edits will show up in the other medium.

If you see cuboids have already been added to the 3D point cloud when you open your task, adjust those cuboids and add additional cuboids as needed.

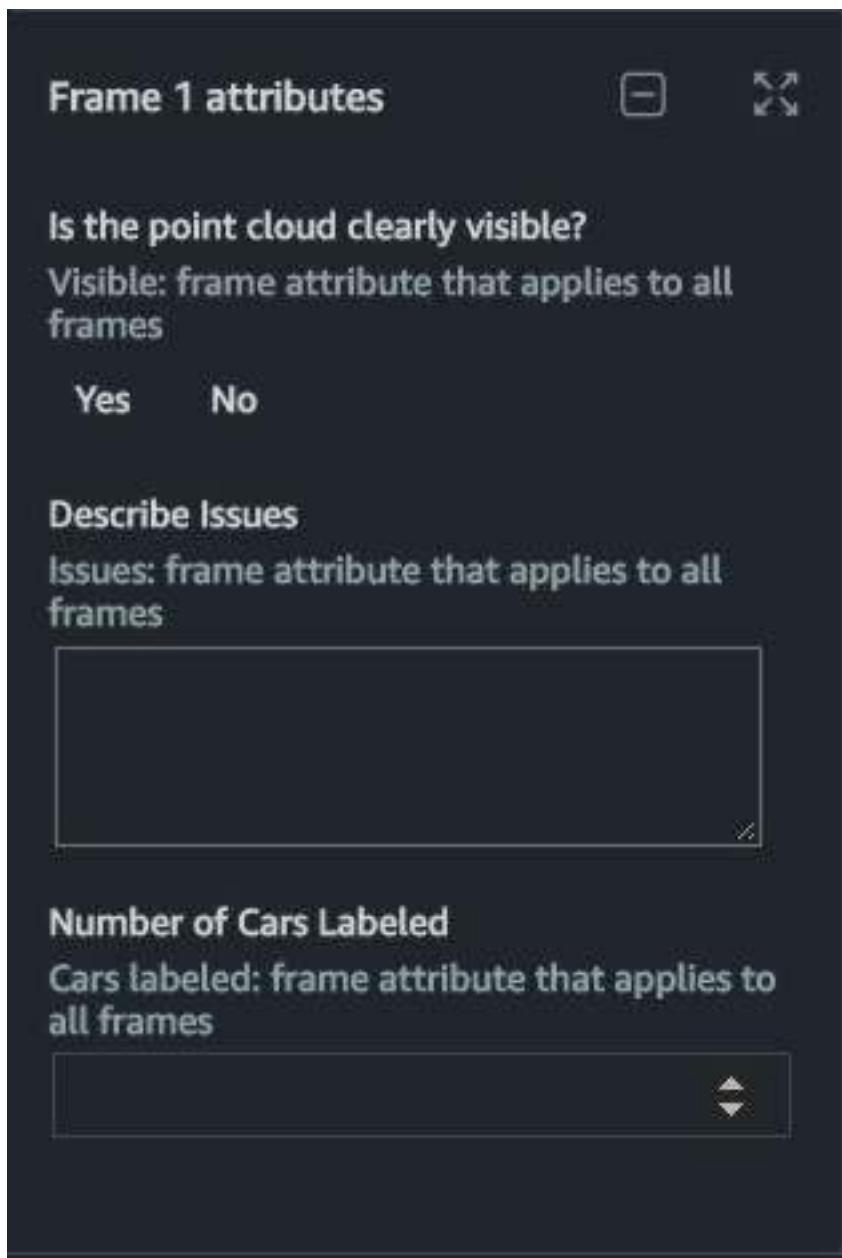
To edit a cuboid, including moving, re-orienting, and changing cuboid dimensions, you must use shortcut keys. You can see a full list of shortcut keys in the **Shortcuts** menu in your UI. The following are important key-combinations that you should become familiar with before starting your labeling task.

Mac Command	Windows Command	Action
Cmd + Drag	Ctrl + Drag	Modify the dimensions of the cuboid.
Option + Drag	Alt + Drag	Move the cuboid.
Shift + Drag	Shift + Drag	Rotate the cuboid.
Option + O	Alt + O	Fit the cuboid tightly around the points it has been drawn around. Before using the option, make

Mac Command	Windows Command	Action
		sure the cuboid fully-surrounds the object of interest.
Option + G	Alt + G	Set the cuboid to the ground.

Individual labels may have one or more label attributes. If a label has a label attribute associated with it, it will appear when you select the downward pointing arrow next to the label from the **Label Id** menu. Fill in required values for all label attributes.

You may see frame attributes under the **Labels** menu. Use these attribute prompts to enter additional information about each frame.



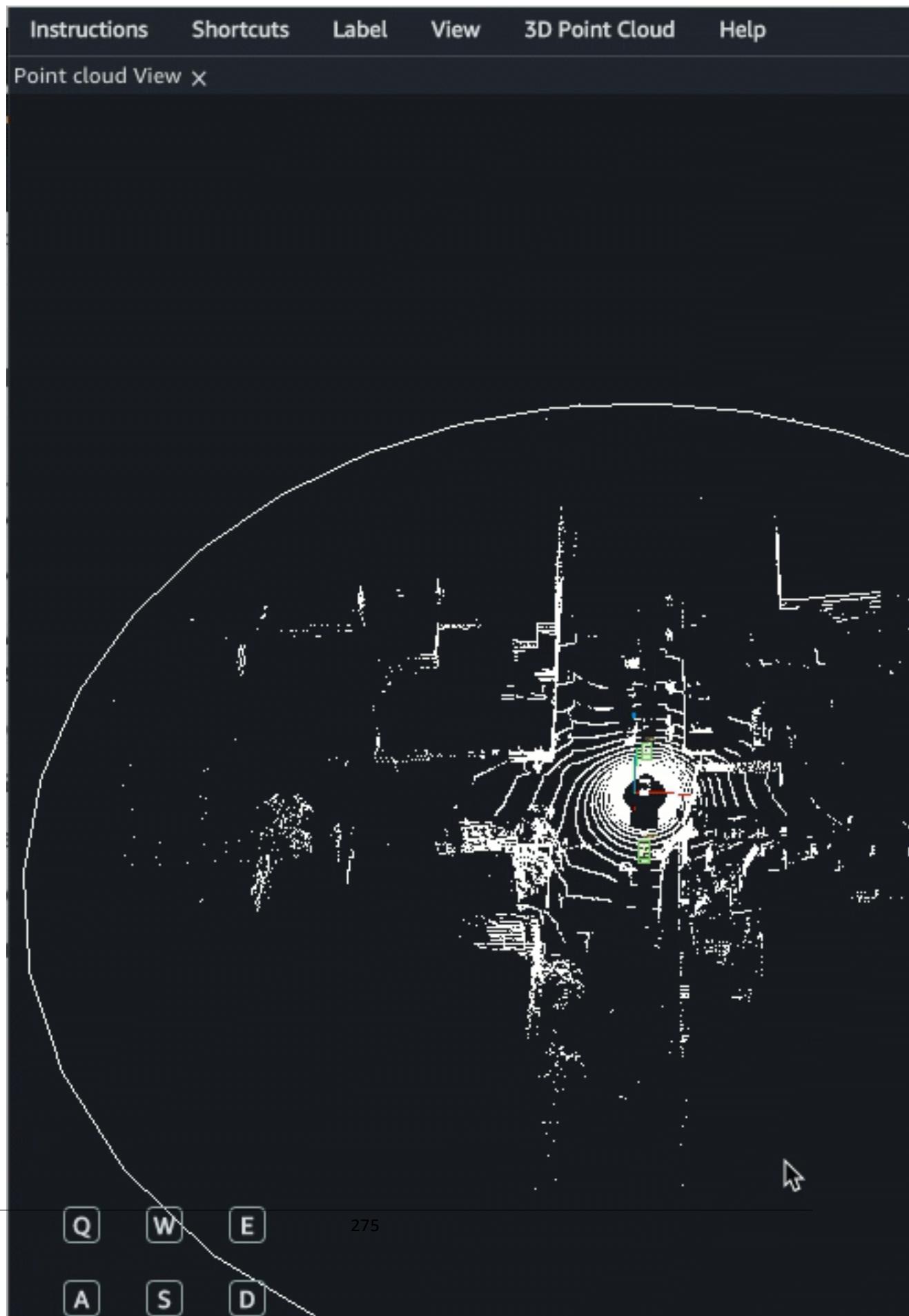
## Navigate the UI

You can navigate in the 3D scene using your keyboard and mouse. You can:

- Double click on specific objects in the point cloud to zoom into them.
- You can use the [ and ] keys on your keyboard to zoom into and move from one label to the next. If no label is selected, when you select [ or ], the UI will zoom into the first label in the **Label Id** list.
- Use a mouse-scroller or trackpad to zoom in and out of the point cloud.
- Use both keyboard arrow keys and Q, E, A, and D keys to move Up, Down, Left, Right. Use keyboard keys W and S to zoom in and out.

Once you place a cuboids in the 3D scene, a side-view will appear with three projected views: top, side, and back. These side-views show points in and around the placed cuboid and help workers refine cuboid boundaries in that area. Workers can zoom in and out of each of those side-views using their mouse.

The following video demonstrates movements around the 3D point cloud and in the side-view.



When you are in the worker UI, you see the following menus:

- **Instructions** – Review these instructions before starting your task.
- **Shortcuts** – Use this menu to view keyboard shortcuts that you can use to navigate the point cloud and use the annotation tools provided.
- **Label** – Use this menu to modify a cuboid. First, select a cuboid, and then choose an option from this menu. This menu includes assistive labeling tools like setting a cuboid to the ground and automatically fitting the cuboid to the object's boundaries.
- **View** – Use this menu to toggle different view options on and off. For example, you can use this menu to add a ground mesh to the point cloud, and to choose the projection of the point cloud.
- **3D Point Cloud** – Use this menu to add additional attributes to the points in the point cloud, such as color, and pixel intensity. Note that these options may not be available.

When you open a task, the move scene icon is on, and you can move around the point cloud using your mouse and the navigation buttons in the point cloud area of the screen. To return to the original view you see when you first opened the task, choose the reset scene icon. Resetting the view will not modify your annotations.

After you select the add cuboid icon, you can add cuboids to the 3D point cloud visualization. Once you've added a cuboid, you can adjust it in the three views (top, side, and front) and in the images (if included).

Hello, chopt@amazon.com

Instructions    **Shortcuts**    Label    View    3D Point Cloud    Help

**Shortcuts**    X    Point cloud View X

Double click on the point cloud to zoom in

Double click on the label ID to zoom in the object on point cloud.

**3D Cuboid Controls**

When you are in Edit Cuboid mode

C Create

V Edit

Cmd + Drag Change dimension

Option + Drag Move cuboid

Option + O Fit label to points

Option + G Set to ground

Shift + Drag Rotate cuboid

[ Previous label

] Next label

Cmd + , Toggle show/hide label

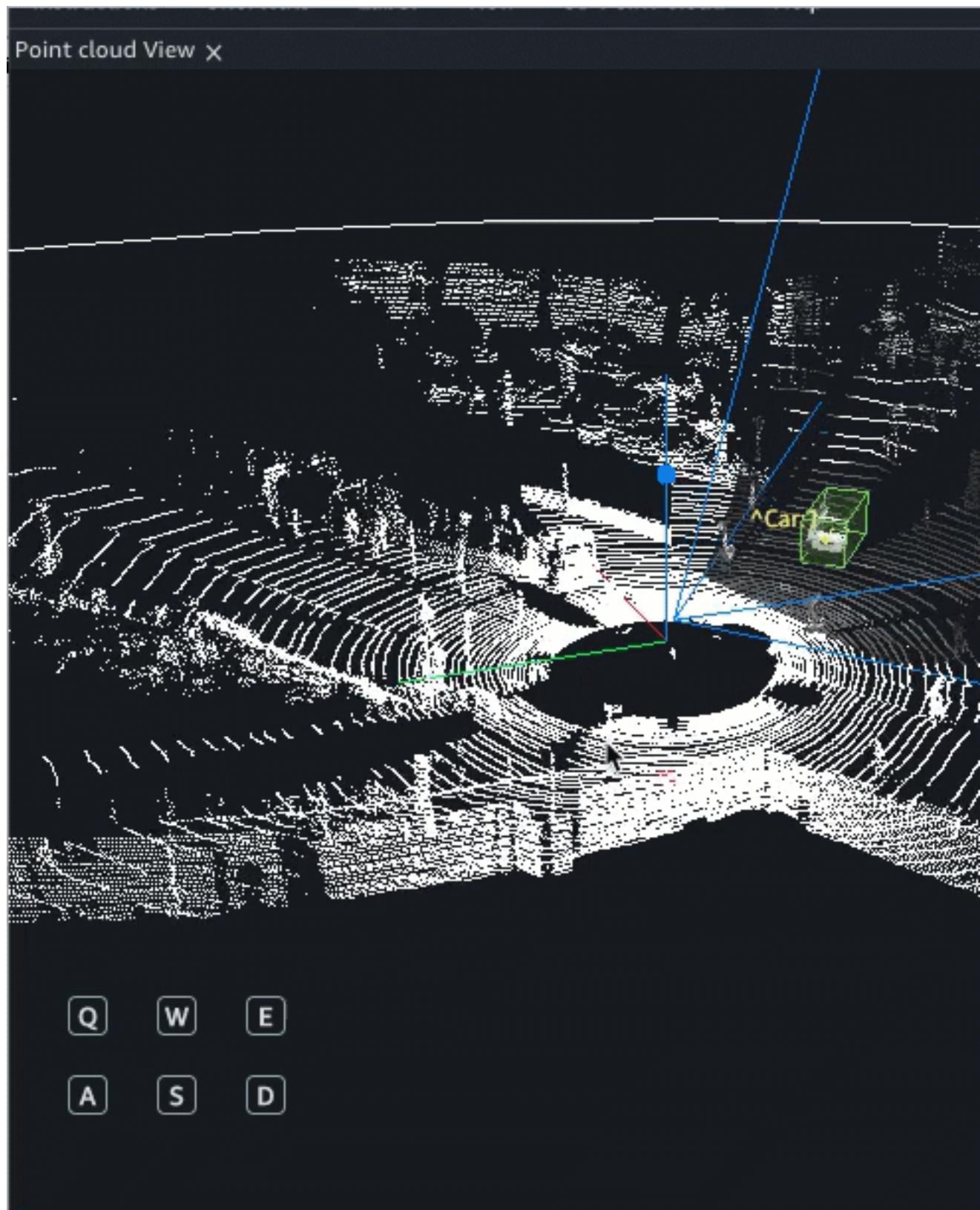
You must choose the move scene icon again to move to another area in the 3D point cloud or image.

To collapse all panels on the right and make the 3D point cloud full-screen, choose the full screen icon.

If camera images are included, you may have the following view options:

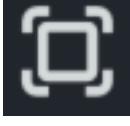
- **C** – View the camera angle on point cloud view.
- **F** – View the frustum, or field of view, of the camera used to capture that image on point cloud view.
- **P** – View the point cloud overlaid on the image.
- **B** – View cuboids in the image.

The following video demonstrates how to use these view options. The **F** option is used to view the field of view of the camera (the gray area), the **C** options shows the direction the camera is facing and angle of the camera (blue lines), and the **B** option is used to view the cuboid.



## Icon Guide

Use this table to learn about the icons you see in your worker task portal.

Icon		Description
	add cuboid	Choose this icon to add a cuboid. Each cuboid you add is associated with the category you chose.
	edit cuboid	Choose this icon to edit a cuboid. After you have added a cuboid, you can edit its dimensions, location, and orientation. After a cuboid is added, it automatically switches to edit cuboid mode.
	ruler	<p>Use this icon to measure distances, in meters, in the point cloud. You may want to use this tool if your instructions ask you to annotate all objects in a given distance from the center of the cuboid or the object used to capture data.</p> <p>When you select this icon, you can place the starting point (first marker) anywhere in the point cloud by selecting it with your mouse. The tool will automatically use interpolation to place a marker on the closest point within threshold distance to the location you select, otherwise the marker will be placed on ground. If you place a starting point by mistake, you can use the Escape key to revert marker placement.</p> <p>After you place the first marker, you see a dotted line and a dynamic label that indicates the distance you have moved away from the first marker. Click somewhere else on the point cloud to place a second marker. When you place the second marker, the dotted line becomes solid, and the distance is set.</p> <p>After you set a distance, you can edit it by selecting either marker. You can delete a ruler by selecting anywhere on the ruler and using the Delete key on your keyboard.</p>
	reset scene	Choose this icon to reset the view of the point cloud, side panels, and if applicable, all images to their original position when the task was first opened.
	move scene	Choose this icon to move the scene. By default, this icon is chosen when you first start a task.

Icon		Description
	full screen	Choose this icon to make the 3D point cloud visualization full screen, and to collapse all side panels.
	show labels	Show labels in the 3D point cloud visualization, and if applicable, in images.
	hide labels	Hide labels in the 3D point cloud visualization, and if applicable, in images.
	delete labels	Delete a label.

## Shortcuts

The shortcuts listed in the **Shortcuts** menu can help you navigate the 3D point cloud and use tools to add and edit cuboids.

Before you start your task, it is recommended that you review the **Shortcuts** menu and become acquainted with these commands. You need to use some of the 3D cuboid controls to edit your cuboid.

## Release, Stop and Resume, and Decline Tasks

When you open the labeling task, three buttons on the top right allow you to decline the task (**Decline task**), release it (**Release task**), and stop and resume it at a later time (**Stop and resume later**). The following list describes what happens when you select one of these options:

- **Decline task:** You should only decline a task if something is wrong with the task, such as an issue with the 3D point cloud, images or the UI. If you decline a task, you will not be able to return to the task.
- **Release Task:** If you release a task, you lose all work done on that task. When the task is released, other workers on your team can pick it up. If enough workers pick up the task, you may not be able to return to it. When you select this button and then select **Confirm**, you are returned to the worker portal. If the task is still available, its status will be **Available**. If other workers pick it up, it will disappear from your portal.
- **Stop and resume later:** You can use the **Stop and resume later** button to stop working and return to the task at a later time. You should use the **Save** button to save your work before you select **Stop and resume later**. When you select this button and then select **Confirm**, you are returned to the worker portal, and the task status is **Stopped**. You can select the same task to resume work on it.

Be aware that the person that creates your labeling tasks specifies a time limit in which all tasks must be completed by. If you do not return to and complete this task within that time limit, it will expire and your work will not be submitted. Contact your administrator for more information.

## Saving Your Work and Submitting

You should periodically save your work. Ground Truth will automatically save your work every 15 minutes.

When you open a task, you must complete your work on it before pressing **Submit**.

## 3D Point Cloud Object Tracking

Use this page to become familiar with the user interface and tools available to complete your 3D point cloud object detection task.

### Topics

- [Your Task \(p. 282\)](#)
- [Navigate the UI \(p. 286\)](#)
- [Bulk Edit Label Category and Frame Attributes \(p. 290\)](#)
- [Icon Guide \(p. 291\)](#)
- [Shortcuts \(p. 292\)](#)
- [Release, Stop and Resume, and Decline Tasks \(p. 292\)](#)
- [Saving Your Work and Submitting \(p. 293\)](#)

### Your Task

When you work on a 3D point cloud object tracking task, you need to select a category from the **Annotations** menu on the right side of your worker portal using the **Label Categories** menu. After you've selected a category, use the add cuboid and fit cuboid tools to fit a cuboid around objects in the 3D point cloud that this category applies to. After you place a cuboid, you can modify its location, dimensions, and orientation directly in the point cloud, and the three panels shown on the right. If you see one or more images in your worker portal, you can also modify cuboids in the images or in the 3D point cloud and the edits will show up in the other medium.

#### Important

If you see cuboids have already been added to the 3D point cloud frames when you open your task, adjust those cuboids and add additional cuboids as needed.

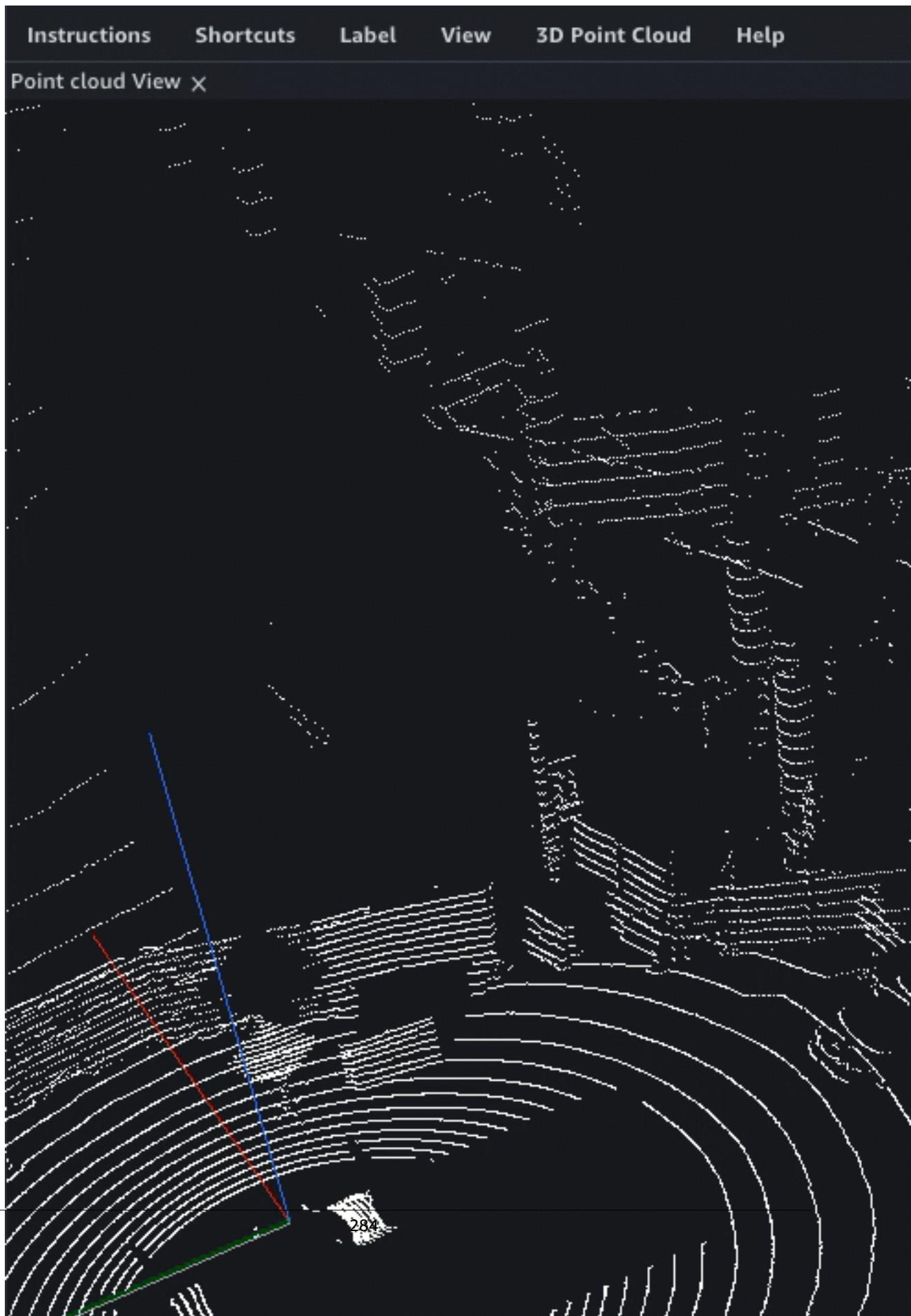
To edit a cuboid, including moving, re-orienting, and changing cuboid dimensions, you must use shortcut keys. You can see a full list of shortcut keys in the **Shortcuts** menu in your UI. The following are important key-combinations that you should become familiar with before starting your labeling task.

Mac Command	Windows Command	Action
Cmd + Drag	Ctrl + Drag	Modify the dimensions of the cuboid.
Option + Drag	Alt + Drag	Move the cuboid.
Shift + Drag	Shift + Drag	Rotate the cuboid.
Option + O	Alt + O	Fit the cuboid tightly around the points it has been drawn around. Before using the option, make sure the cuboid fully-surrounds the object of interest.
Option + G	Alt + G	Set the cuboid to the ground.

When you open your task, two frames will be loaded. If your task includes more than two frames, you need to use the navigation bar in the lower-left corner, or the load frames icon to load additional frames. You should annotate and adjust labels in all frames before submitting.

After you fit a cuboid tightly around the boundaries of an object, navigate to another frame using the navigation bar in the lower-right corner of the UI. If that same object has moved to a new location, add another cuboid and fit it tightly around the boundaries of the object. Each time you manually add a cuboid, you see the frame sequence bar in the lower-left corner of the screen turn red where that frame is located temporally in the sequence.

Your UI automatically infers the location of that object in all other frames after you've placed a cuboid. This is called *interpolation*. You can see the movement of that object, and the inferred and manually created cuboids using the arrows. Adjust inferred cuboids as needed. The following video demonstrates how to navigate between frames. The following video shows how, if you add a cuboid in one frame, and then adjust it in another, your UI will automatically infer the location of the cuboid in all of the frames in-between.



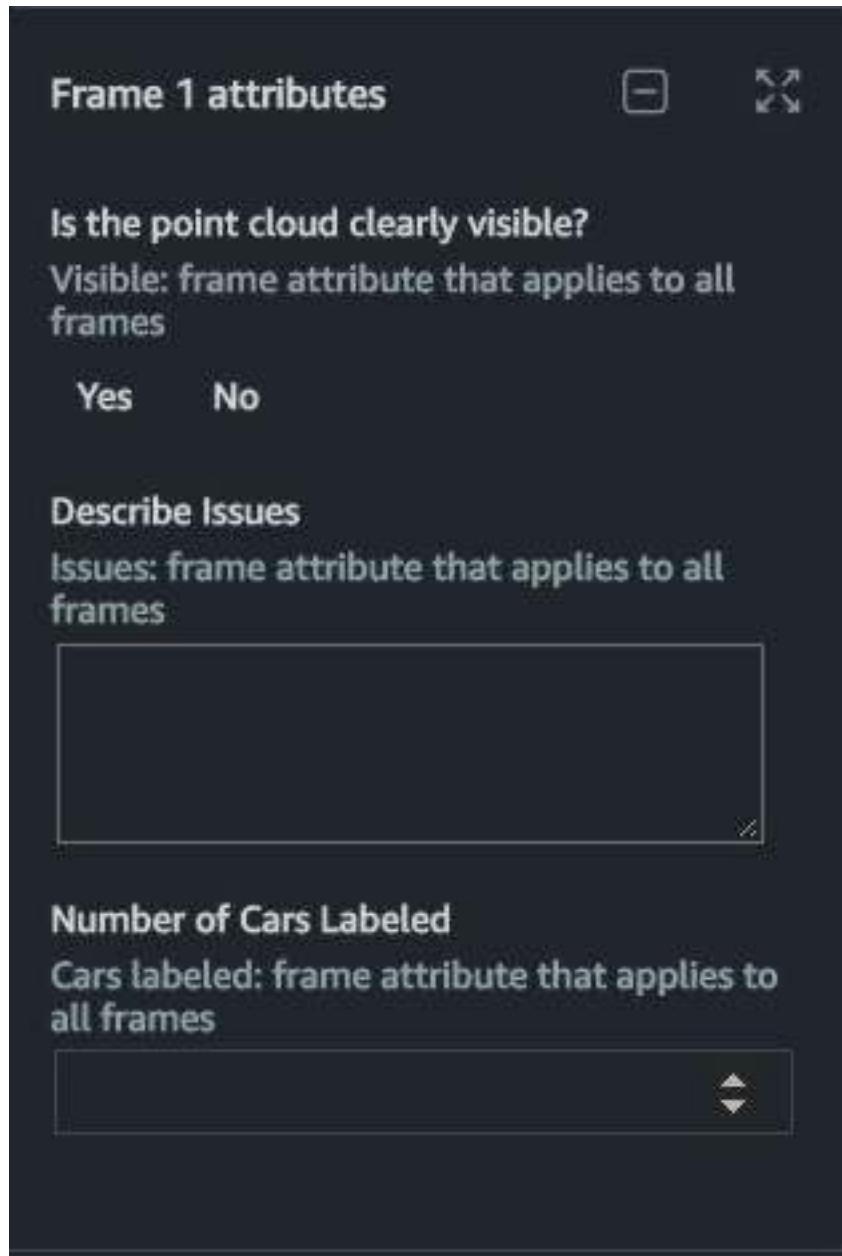
**Tip**

You can turn off the automatic cuboid interpolation across frames using the 3D Point Cloud menu item. Select **3D Point Cloud** from the top-menu, and then select **Interpolate Cuboids Across Frames**. This will uncheck this option and stop cuboid interpolation. You can reselect this item to turn cuboid interpolation back on.

Turning cuboid interpolation off will not impact cuboids that have already been interpolated across frames.

Individual labels may have one or more label attributes. If a label has a label attribute associated with it, it will appear when you select the downward pointing arrow next to the label from the **Label Id** menu. Fill in required values for all label attributes.

You may see frame attributes under the **Label Id** menu. These attributes will appear on each frame in your task. Use these attribute prompts to enter additional information about each frame.



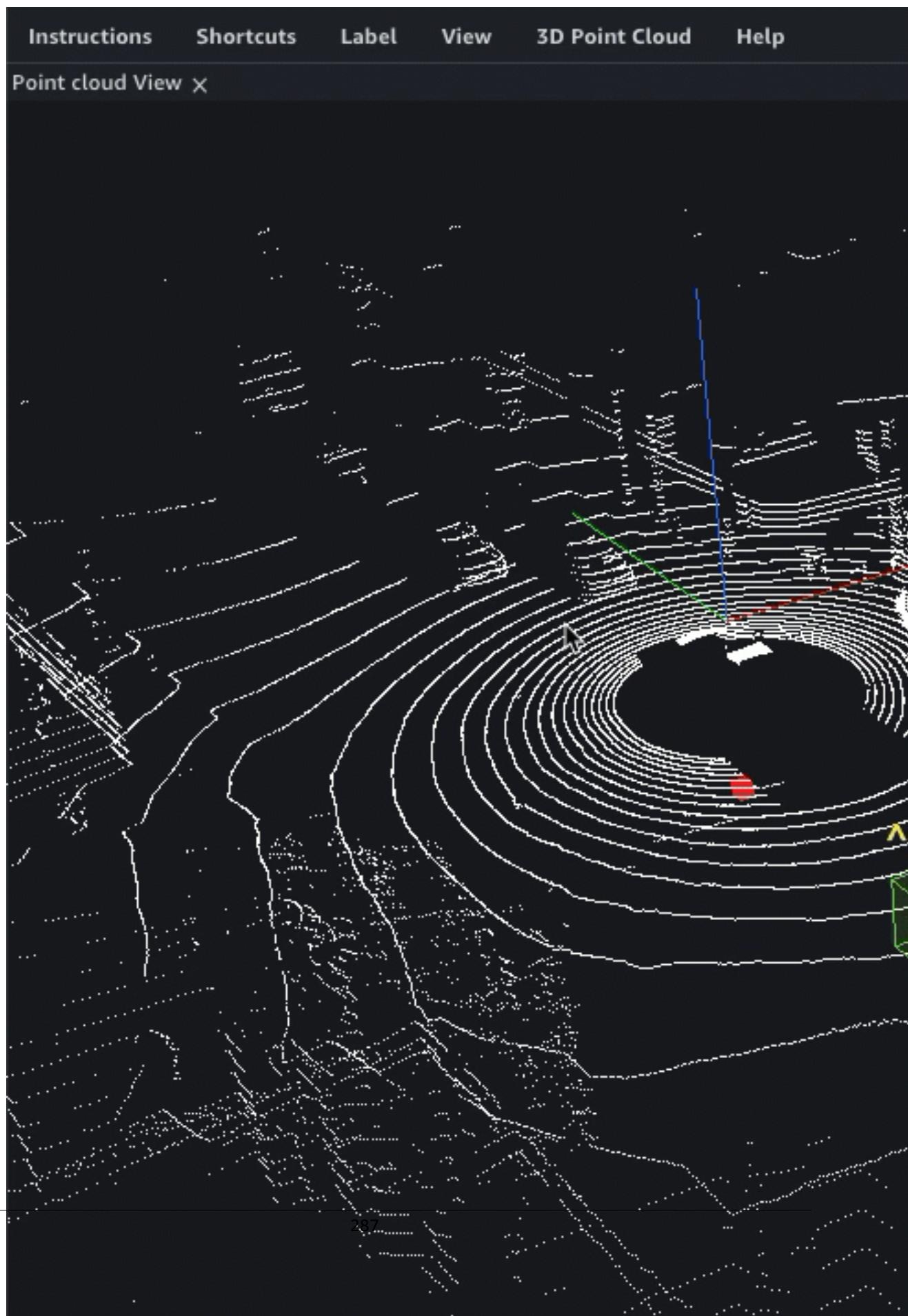
## Navigate the UI

You can navigate in the 3D scene using your keyboard and mouse. You can:

- Double click on specific objects in the point cloud to zoom into them.
- You can use the [ and ] keys on your keyboard to zoom into and move from one label to the next. If no label is selected, when you select [ or ], the UI will zoom into the first label in the **Label Id** list.
- Use a mouse-scroller or trackpad to zoom in and out of the point cloud.
- Use both keyboard arrow keys and Q, E, A, and D keys to move Up, Down, Left, Right. Use keyboard keys W and S to zoom in and out.

Once you place a cuboids in the 3D scene, a side-view will appear with three projected views: top, side, and back. These side-views show points in and around the placed cuboid and help workers refine cuboid boundaries in that area. Workers can zoom in and out of each of those side-views using their mouse.

The following video demonstrates movements around the 3D point cloud and in the side-view.



When you are in the worker UI, you see the following menus:

- **Instructions** – Review these instructions before starting your task.
- **Shortcuts** – Use this menu to view keyboard shortcuts that you can use to navigate the point cloud and use the annotation tools provided.
- **Label** – Use this menu to modify a cuboid. First, select a cuboid, and then choose an option from this menu. This menu includes assistive labeling tools like setting a cuboid to the ground and automatically fitting the cuboid to the object's boundaries.
- **View** – Use this menu to toggle different view options on and off. For example, you can use this menu to add a ground mesh to the point cloud, and to choose the projection of the point cloud.
- **3D Point Cloud** – Use this menu to add additional attributes to the points in the point cloud, such as color, and pixel intensity. Note that these options may not be available.

When you open a task, the move scene icon is on, and you can move around the point cloud using your mouse and the navigation buttons in the point cloud area of the screen. To return to the original view you see when you first opened the task, choose the reset scene icon.

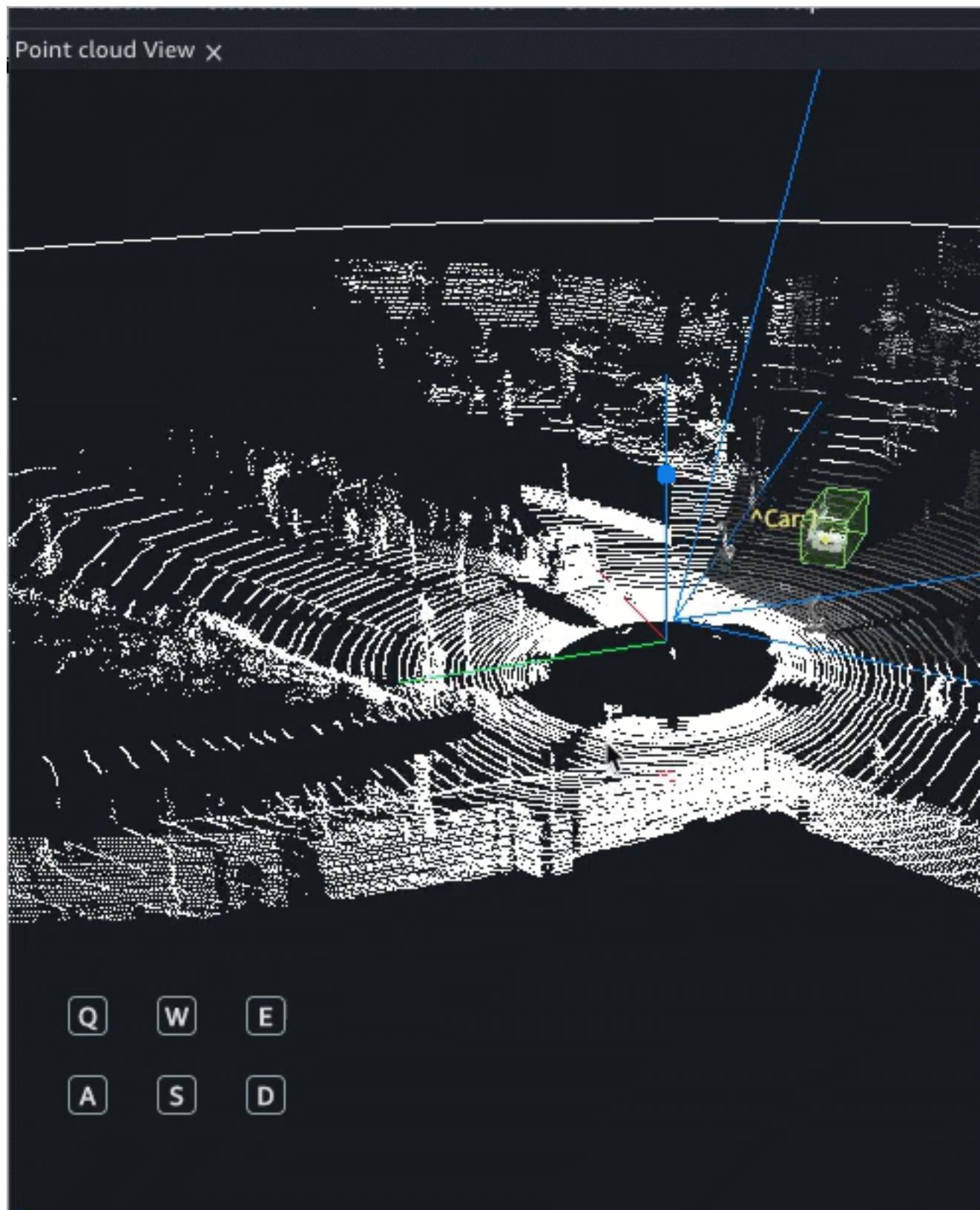
After you select the add cuboid icon, you can add cuboids to the point cloud and images (if included). You must select the move scene icon again to move to another area in the 3D point cloud or image.

To collapse all panels on the right and make the 3D point cloud full-screen, choose the full screen icon.

If camera images are included, you may have the following view options:

- **C** – View the camera angle on point cloud view.
- **F** – View the frustum, or field of view, of the camera used to capture that image on point cloud view.
- **P** – View the point cloud overlaid on the image.
- **B** – View cuboids in the image.

The following video demonstrates how to use these view options. The **F** option is used to view the field of view of the camera (the gray area), the **C** options shows the direction the camera is facing and angle of the camera (blue lines), and the **B** option is used to view the cuboid.



## Delete Cuboids

You can select a cuboid or label ID and:

- Delete an individual cuboid in the current frame you are viewing.
- Delete all cuboids with that label ID before or after the frame you are viewing.
- Delete all cuboids with that label ID in all frames.

A common use-case for cuboid deletion is if the object leaves the scene.

You can use one or more of these options to delete both manually placed and interpolated cuboids with the same label ID.

- To delete all cuboids before or after the frame you are currently on, select the cuboid, select the **Label** menu item at the top of the UI and then select one of **Delete in previous frames** or **Delete in next frames**. Use the Shortcuts menu to see the shortcut keys you can use for these options.
- To delete a label in all frames, select **Delete in all frames** from the **Labels** menu, or use the shortcut **Shift + Delete** on your keyboard.
- To delete an individual cuboid from a single frame, select the cuboid and either select the trashcan icon (trash bin icon) next to that label ID in the **Label ID** sidebar on the right or use the Delete key on your keyboard to delete that cuboid.

If you have manually placed more than one cuboid with the same label in different frames, when you delete one of the manually placed cuboids, all interpolated cuboids adjust. This adjustment happens because the UI uses manually placed cuboids as anchor points when calculating the location of interpolated cuboid. When you remove one of these anchor points, the UI must recalculate the position of interpolated cuboids.

If you delete a cuboid from a frame, but later decide that you want to get it back, you can use the **Duplicate to previous frames** or **Duplicate to next frames** options in the **Label** menu to copy the cuboid into all the previous or all of the following frames, respectively.

## Bulk Edit Label Category and Frame Attributes

You can bulk edit label attributes and frame attributes.

When you bulk edit an attribute, you specify one or more ranges of frames that you want to apply the edit to. The attribute you select is edited in all frames in that range, including the start and end frames you specify. When you bulk edit label attributes, the range you specify *must* contain the label that the label attribute is attached to. If you specify frames that do not contain this label, you will receive an error.

To bulk edit an attribute you *must* specify the desired value for the attribute first. For example, if you want to change an attribute from *Yes* to *No*, you must select *No*, and then perform the bulk edit.

You can also specify a new value for an attribute that has not been filled in and then use the bulk edit feature to fill in that value in multiple frames. To do this, select the desired value for the attribute and complete the following procedure.

### To bulk edit a label or attribute:

1. Use your mouse to right click the attribute you want to bulk edit.
2. Specify the range of frames you want to apply the bulk edit to using a dash (-) in the text box. For example, if you want to apply the edit to frames one through ten, enter 1-10. If you want to apply the edit to frames two to five, eight to ten and twenty enter 2-5, 8-10, 20.
3. Select **Confirm**.

If you get an error message, verify that you entered a valid range and that the label associated with the label attribute you are editing (if applicable) exists in all frames specified.

You can quickly add a label to all previous or subsequent frames using the **Duplicate to previous frames** and **Duplicate to next frames** options in the **Label** menu at the top of your screen.

### Icon Guide

Use this table to learn about the icons you see in your worker task portal.

Icon		Description
	add cuboid	Choose this icon to add a cuboid. Each cuboid you add is associated with the category you chose.
	edit cuboid	Choose this icon to edit a cuboid. After you add a cuboid, you can edit its dimensions, location, and orientation. After a cuboid is added, it automatically switches to edit cuboid mode.
	ruler	<p>Use this icon to measure distances, in meters, in the point cloud. You may want to use this tool if your instructions ask you to annotate all objects in a given distance from the center of the cuboid or the object used to capture data.</p> <p>When you select this icon, you can place the starting point (first marker) anywhere in the point cloud by selecting it with your mouse. The tool will automatically use interpolation to place a marker on the closest point within threshold distance to the location you select, otherwise the marker will be placed on ground. If you place a starting point by mistake, you can use the Escape key to revert marker placement.</p> <p>After you place the first marker, you see a dotted line and a dynamic label that indicates the distance you have moved away from the first marker. Click somewhere else on the point cloud to place a second marker. When you place the second marker, the dotted line becomes solid, and the distance is set.</p> <p>After you set a distance, you can edit it by selecting either marker. You can delete a ruler by selecting anywhere on the ruler and using the Delete key on your keyboard.</p>
	reset scene	Choose this icon to reset the view of the point cloud, side panels, and if applicable, all images to their original position when the task was first opened.
	move scene	Choose this icon to move the scene. By default, this icon is chosen when you first start a task.

Icon		Description
	full screen	Choose this icon to make the 3D point cloud visualization full screen and to collapse all side panels.
	load frames	Choose this icon to load additional frames.
	hide labels	Hide labels in the 3D point cloud visualization, and if applicable, in images.
	show labels	Show labels in the 3D point cloud visualization, and if applicable, in images.
	delete labels	Delete a label. This option can only be used to delete labels you have manually created or adjusted.

## Shortcuts

The shortcuts listed in the **Shortcuts** menu can help you navigate the 3D point cloud and use tools to add and edit cuboids.

Before you start your task, it is recommended that you review the **Shortcuts** menu and become acquainted with these commands. You need to use some of the 3D cuboid controls to edit your cuboid.

## Release, Stop and Resume, and Decline Tasks

When you open the labeling task, three buttons on the top right allow you to decline the task (**Decline task**), release it (**Release task**), and stop and resume it at a later time (**Stop and resume later**). The following list describes what happens when you select one of these options:

- **Decline task:** You should only decline a task if something is wrong with the task, such as an issue with the 3D point clouds, images or the UI. If you decline a task, you will not be able to return to the task.
- **Release Task:** Use this option to release a task and allow others to work on it. When you release a task, you lose all work done on that task and other workers on your team can pick it up. If enough workers pick up the task, you may not be able to return to it. When you select this button and then select **Confirm**, you are returned to the worker portal. If the task is still available, its status will be **Available**. If other workers pick it up, it will disappear from your portal.
- **Stop and resume later:** You can use the **Stop and resume later** button to stop working and return to the task at a later time. You should use the **Save** button to save your work before you select **Stop and resume later**. When you select this button and then select **Confirm**, you are returned to the worker portal, and the task status is **Stopped**. You can select the same task to resume work on it.

Be aware that the person that creates your labeling tasks specifies a time limit in which all tasks must be completed by. If you do not return to and complete this task within that time limit, it will expire and your work will not be submitted. Contact your administrator for more information.

## Saving Your Work and Submitting

You should periodically save your work. Ground Truth will automatically save your work every 15 minutes.

When you open a task, you must complete your work on it before pressing **Submit**.

# Verify and Adjust Labels

When the labels on a dataset need to be validated, Amazon SageMaker Ground Truth provides functionality to have workers verify that labels are correct or to adjust previous labels.

These types of jobs fall into two distinct categories:

- *Label verification* — Workers indicate if the existing labels are correct, or rate their quality, and can add comments to explain their reasoning. Workers will not be able to modify or adjust labels.

If you create a 3D point cloud or video frame label adjustment or verification job, you can choose to make label category attributes (not supported for 3D point cloud semantic segmentation) and frame attributes editable by workers.

- *Label adjustment* — Workers adjust prior annotations and, if applicable, label category and frame attributes to correct them.

The following Ground Truth [built-in task types](#) support adjustment and verification labeling jobs:

- Bounding box
- Semantic segmentation
- 3D point cloud object detection, 3D point cloud object tracking, and 3D point cloud semantic segmentation
- All video frame object detection and video frame object tracking task types — bounding box, polyline, polygon and keypoint

### Tip

For 3D point cloud and video frame labeling verification jobs, it is recommended that you add new label category attributes or frame attributes to the labeling job. Workers can use these attributes to verify individual labels or the entire frame. To learn more about label category and frame attributes, see [Worker User Interface \(UI\) \(p. 260\)](#) for 3D point cloud and [Worker User Interface \(UI\) \(p. 212\)](#) for video frame.

You can start a label verification and adjustment jobs using the SageMaker console or the API.

### Topics

- [Requirements to Create Verification and Adjustment Labeling Jobs \(p. 294\)](#)
- [Create a Label Verification Job \(Console\) \(p. 294\)](#)
- [Create a Label Adjustment Job \(Console\) \(p. 296\)](#)
- [Start a Label Verification or Adjustment Job \(API\) \(p. 297\)](#)
- [Label Verification and Adjustment Data in the Output Manifest \(p. 299\)](#)
- [Cautions and Considerations \(p. 299\)](#)

## Requirements to Create Verification and Adjustment Labeling Jobs

To create a label verification or adjustment job, the following criteria must be satisfied.

- For non streaming labeling jobs: The input manifest file you use must contain the label attribute name (`LabelAttributeName`) of the labels that you want adjusted. When you chain a successfully completed labeling job, the output manifest file is used as the input manifest file for the new, chained job. To learn more about the format of the output manifest file Ground Truth produces for each task type, see [Output Data \(p. 404\)](#).

For streaming labeling jobs: The Amazon SNS message you sent to the Amazon SNS input topic of the adjustment or verification labeling job must contain the label attribute name of the labels you want adjusted or verified. To see an example of how you can create an adjustment or verification labeling job with streaming labeling jobs, see this [Jupyter Notebook example](#) in GitHub.

- The task type of the verification or adjustment labeling job must be the same as the task type of the original job unless you are using the [Image Label Verification \(p. 187\)](#) task type to verify bounding box or semantic segmentation image labels. See the next bullet point for more details about the video frame task type requirements.
- For video frame annotation verification and adjustment jobs, you must use the same annotation task type used to create the annotations from the previous labeling job. For example, if you create a video frame object detection job to have workers draw bounding boxes around objects, and then you create a video object detection adjustment job, you must specify *bounding boxes* as the annotation task type. To learn more video frame annotation task types, see [Task Types \(p. 211\)](#).
- The task type you select for the adjustment or verification labeling job must support an audit workflow. The following Ground Truth [built-in task types](#) support adjustment and verification labeling jobs: bounding box, semantic segmentation, 3D point cloud object detection, 3D point cloud object tracking, and 3D point cloud semantic segmentation, and all video frame object detection and video frame object tracking task types — bounding box, polyline, polygon and keypoint.

## Create a Label Verification Job (Console)

Bounding box and semantic segmentation labeling jobs are created by choosing the **Label verification** task type in the console. To create a verification job for 3D point cloud and video frame task types, you must choose the same task type as the original labeling job and choose to display existing labels. Use one of the following sections to create a label verification job for your task type.

### Topics

- [Create an Image Label Verification Job \(Console\) \(p. 294\)](#)
- [Create a Point Cloud or Video Frame Label Verification Job \(Console\) \(p. 295\)](#)

### Create an Image Label Verification Job (Console)

Use the following procedure to create a bounding box or semantic segmentation verification job using the console. This procedure assumes that you have already created a bounding box or semantic segmentation labeling job and its status is Complete. This is the labeling job that produces the labels you want verified.

#### To create an image label verification job:

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/> and choose **Labeling jobs**.
2. Start a new labeling job by [chaining \(p. 436\)](#) a prior job or start from scratch, specifying an input manifest that contains labeled data objects.

3. In the **Task type** pane, select **Label verification**.
4. Choose **Next**.
5. In the **Workers** section, choose the type of workforce you would like to use. For more details about your workforce options see [Create and Manage Workforces \(p. 456\)](#).
6. (Optional) After you've selected your workforce, specify the **Task timeout** and **Task expiration time**.
7. In the **Existing-labels display options** pane, the system shows the available label attribute names in your manifest. Choose the label attribute name that identifies the labels that you want workers to verify. Ground Truth tries to detect and populate these values by analyzing the manifest, but you might need to set the correct value.
8. Use the instructions areas of the tool designer to provide context about what the previous labelers were asked to do and what the current verifiers need to check.

You can add new labels that workers choose from to verify labels. For example, you can ask workers to verify the image quality, and provide the labels *Clear* and *Blurry*. Workers will also have the option to add a comment to explain their selection.

9. Choose **See preview** to check that the tool is displaying the prior labels correctly and presents the label verification task clearly.
10. Select **Create**. This will create and start your labeling job.

## Create a Point Cloud or Video Frame Label Verification Job (Console)

Use the following procedure to create a 3D point cloud or video frame verification job using the console. This procedure assumes that you have already created a labeling job using the task type that produces the types of labels you want to be verified and its status is Complete.

### To create an image label verification job:

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/> and choose **Labeling jobs**.
2. Start a new labeling job by [chaining \(p. 436\)](#) a prior job or start from scratch, specifying an input manifest that contains labeled data objects.
3. In the **Task type** pane, select the same task type as the labeling job that you chained. For example, if the original labeling job was a video frame object detection keypoint labeling job, select that task type.
4. Choose **Next**.
5. In the **Workers** section, choose the type of workforce you would like to use. For more details about your workforce options see [Create and Manage Workforces \(p. 456\)](#).
6. (Optional) After you've selected your workforce, specify the **Task timeout** and **Task expiration time**.
7. Toggle on the switch next to **Display existing labels**.
8. Select **Verification**.
9. For **Label attribute name**, choose the name from your manifest that corresponds to the labels that you want to display for verification. You will only see label attribute names for labels that match the task type you selected on the previous screen. Ground Truth tries to detect and populate these values by analyzing the manifest, but you might need to set the correct value.
10. Use the instructions areas of the tool designer to provide context about what the previous labelers were asked to do and what the current verifiers need to check.

You cannot modify or add new labels. You can remove, modify and add new label category attributes or frame attributes. It is recommended that you add new label category attributes or frame attributes to the labeling job. Workers can use these attribute to verify individual labels or the entire frame.

By default, preexisting label category attributes and frame attributes will not be editable by workers. If you want to make a label category or frame attribute editable, select the **Allow workers to edit this attribute** check box for that attribute.

To learn more about label category and frame attributes, see [Worker User Interface \(UI\) \(p. 260\)](#) for 3D point cloud and [Worker User Interface \(UI\) \(p. 212\)](#) for video frame.

11. Choose **See preview** to check that the tool is displaying the prior labels correctly and presents the label verification task clearly.
12. Select **Create**. This will create and start your labeling job.

## Create a Label Adjustment Job (Console)

Use one of the following sections to create a label verification job for your task type.

### Topics

- [Create an Image Label Adjustment Job \(Console\) \(p. 296\)](#)
- [Create a Point Cloud or Video Frame Label Adjustment Job \(Console\) \(p. 297\)](#)

### Create an Image Label Adjustment Job (Console)

Use the following procedure to create a bounding box or semantic segmentation adjustment labeling job using the console. This procedure assumes that you have already created a bounding box or semantic segmentation labeling job and its status is Complete. This the labeling job that produces the labels you want adjusted.

#### To create an image label adjustment job (console)

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/> and choose **Labeling jobs**.
2. Start a new labeling job by [chaining \(p. 436\)](#) a prior job or start from scratch, specifying an input manifest that contains labeled data objects.
3. Choose the same task type as the original labeling job.
4. Choose **Next**.
5. In the **Workers** section, choose the type of workforce you would like to use. For more details about your workforce options see [Create and Manage Workforces \(p. 456\)](#).
6. (Optional) After you've selected your workforce, specify the **Task timeout** and **Task expiration time**.
7. Expand **Existing-labels display options** by selecting the arrow next to the title.
8. Check the box next to **I want to display existing labels from the dataset for this job**.
9. For **Label attribute name**, choose the name from your manifest that corresponds to the labels that you want to display for adjustment. You will only see label attribute names for labels that match the task type you selected on the previous screen. Ground Truth tries to detect and populate these values by analyzing the manifest, but you might need to set the correct value.
10. Use the instructions areas of the tool designer to provide context about what the previous labelers were tasked with doing and what the current verifiers need to check and adjust.
11. Choose **See preview** to check that the tool shows the prior labels correctly and presents the task clearly.
12. Select **Create**. This will create and start your labeling job.

## Create a Point Cloud or Video Frame Label Adjustment Job (Console)

Use the following procedure to create a 3D point cloud or video frame adjustment job using the console. This procedure assumes that you have already created a labeling job using the task type that produces the types of labels you want to be verified and its status is Complete.

### To create a 3D point cloud or video frame label adjustment job (console)

1. Open the SageMaker console: <https://console.amazonaws.cn/sagemaker/> and choose **Labeling jobs**.
2. Start a new labeling job by [chaining \(p. 436\)](#) a prior job or start from scratch, specifying an input manifest that contains labeled data objects.
3. Choose the same task type as the original labeling job.
4. Toggle on the switch next to **Display existing labels**.
5. Select **Adjustment**.
6. For **Label attribute name**, choose the name from your manifest that corresponds to the labels that you want to display for adjustment. You will only see label attribute names for labels that match the task type you selected on the previous screen. Ground Truth tries to detect and populate these values by analyzing the manifest, but you might need to set the correct value.
7. Use the instructions areas of the tool designer to provide context about what the previous labelers were asked to do and what the current adjusters need to check.

You cannot remove or modify existing labels but you can add new labels. You can remove, modify and add new label category attributes or frame attributes.

Be default, preexisting label category attributes and frame attributes will be editable by workers. If you want to make a label category or frame attribute uneditable, deselect the **Allow workers to edit this attribute** check box for that attribute.

To learn more about label category and frame attributes, see [Worker User Interface \(UI\) \(p. 260\)](#) for 3D point cloud and [Worker User Interface \(UI\) \(p. 212\)](#) for video frame.

8. Choose **See preview** to check that the tool shows the prior labels correctly and presents the task clearly.
9. Select **Create**. This will create and start your labeling job.

## Start a Label Verification or Adjustment Job (API)

Start a label verification or adjustment job by chaining a successfully completed job or starting a new job from scratch using the `CreateLabelingJob` operation. The procedure is almost the same as setting up a new labeling job with `CreateLabelingJob`, with a few modifications. Use the following sections to learn what modifications are required to chain a labeling job to create an adjustment or verification labeling job.

When you create an adjustment or verification labeling job using the Ground Truth API, you *must* use a different `LabelAttributeName` than the original labeling job. The original labeling job is the job used to create the labels you want adjusted or verified.

#### Important

The label category configuration file you identify for an adjustment or verification job in `LabelCategoryConfigS3Uri` of `CreateLabelingJob` must contain the same labels used in the original labeling job. You can add new labels. For 3D point cloud and video frame jobs, you can add new label category and frame attributes to the label category configuration file.

### Bounding Box and Semantic Segmentation

To create a bounding box or semantic segmentation label verification or adjustment job, use the following guidelines to specify API attributes for the `CreateLabelingJob` operation.

- Use the [LabelAttributeName](#) parameter to specify the output label name that you want to use for verified or adjusted labels. You must use a different [LabelAttributeName](#) than the one used for the original labeling job.
- If you are chaining the job, the labels from the previous labeling job to be adjusted or verified will be specified in the custom UI template. To learn how to create a custom template, see [Create Custom Worker Task Templates \(p. 2368\)](#).

Identify the location of the UI template in the [UiTemplateS3Uri](#) parameter. SageMaker provides widgets that you can use in your custom template to display old labels. Use the [initial-value](#) attribute in one of the following crowd elements to extract the labels that need verification or adjustment and include them in your task template:

- [crowd-semantic-segmentation \(p. 541\)](#)—Use this crowd element in your custom UI task template to specify semantic segmentation labels that need to be verified or adjusted.
- [crowd-bounding-box \(p. 488\)](#)—Use this crowd element in your custom UI task template to specify bounding box labels that need to be verified or adjusted.
- The [LabelCategoryConfigS3Uri](#) parameter must contain the same label categories as the previous labeling job.
- Use the bounding box or semantic segmentation adjustment or verification lambda ARNs for [PreHumanTaskLambdaArn](#) and [AnnotationConsolidationLambdaArn](#):
  - For bounding box, the adjustment labeling job lambda function ARNs end with [AdjustmentBoundingBox](#) and the verification lambda function ARNs end with [VerificationBoundingBox](#).
  - For semantic segmentation, the adjustment labeling job lambda function ARNs end with [AdjustmentSemanticSegmentation](#) and the verification lambda function ARNs end with [VerificationSemanticSegmentation](#).

### 3D Point Cloud and Video Frame

- Use the [LabelAttributeName](#) parameter to specify the output label name that you want to use for verified or adjusted labels. You must use a different [LabelAttributeName](#) than the one used for the original labeling job.
- You must use the human task UI Amazon Resource Name (ARN) ([HumanTaskUiArn](#)) used for the original labeling job. To see supported ARNs, see [HumanTaskUiArn](#).
- In the label category configuration file, you must specify the label attribute name ([LabelAttributeName](#)) of the previous labeling job that you use to create the adjustment or verification labeling job in the [auditLabelAttributeName](#) parameter.
- You specify whether your labeling job is a *verification* or *adjustment* labeling job using the [editsAllowed](#) parameter in your label category configuration file identified by the [LabelCategoryConfigS3Uri](#) parameter.
  - For *verification* labeling jobs, you must use the [editsAllowed](#) parameter to specify that all labels cannot be modified. [editsAllowed](#) must be set to "none" in each entry in [labels](#). Optionally, you can specify whether or not label categories attributes and frame attributes can be adjusted by workers.
  - Optionally, for *adjustment* labeling jobs, you can use the [editsAllowed](#) parameter to specify labels, label category attributes, and frame attributes that can or cannot be modified by workers. If you do not use this parameter, all labels, label category attributes, and frame attributes will be adjustable.

To learn more about the [editsAllowed](#) parameter and configuring your label category configuration file, see [Label Category Configuration File Schema \(p. 349\)](#).

- Use the 3D point cloud or video frame adjustment lambda ARNs for [PreHumanTaskLambdaArn](#) and [AnnotationConsolidationLambdaArn](#) for both adjustment and verification labeling jobs:

- For 3D point clouds, the adjustment and verification labeling job lambda function ARNs end with `Adjustment3DPointCloudSemanticSegmentation`, `Adjustment3DPointCloudObjectTracking`, and `Adjustment3DPointCloudObjectDetection` for 3D point cloud semantic segmentation, object detection, and object tracking respectively.
- For video frames, the adjustment and verification labeling job lambda function ARNs end with `AdjustmentVideoObjectDetection` and `AdjustmentVideoObjectTracking` for video frame object detection and object tracking respectively.

Ground Truth stores the output data from a label verification or adjustment job in the S3 bucket that you specified in the `S3OutputPath` parameter of the `CreateLabelingJob` operation. For more information about the output data from a label verification or adjustment labeling job, see [Label Verification and Adjustment Data in the Output Manifest \(p. 299\)](#).

## Label Verification and Adjustment Data in the Output Manifest

Amazon SageMaker Ground Truth writes label verification data to the output manifest within the metadata for the label. It adds two properties to the metadata:

- A `type` property, with a value of "groundtruth/label-verification".
- A `worker-feedback` property, with an array of `comment` values. This property is added when the worker enters comments. If there are no comments, the field doesn't appear.

The following example output manifest shows how label verification data appears:

```
{  
    "source-ref": "S3 bucket location",  
    "verify-bounding-box": "1",  
    "verify-bounding-box-metadata":  
    {  
        "class-name": "bad",  
        "confidence": 0.93,  
        "type": "groundtruth/label-verification",  
        "job-name": "verify-bounding-boxes",  
        "human-annotated": "yes",  
        "creation-date": "2018-10-18T22:18:13.527256",  
        "worker-feedback": [  
            {"comment": "The bounding box on the bird is too wide on the right side."},  
            {"comment": "The bird on the upper right is not labeled."}  
        ]  
    }  
}
```

The worker output of adjustment tasks resembles the worker output of the original task, except that it contains the adjusted values and an `adjustment-status` property with the value of `adjusted` or `unadjusted` to indicate whether an adjustment was made.

For more examples of the output of different tasks, see [Output Data \(p. 404\)](#).

## Cautions and Considerations

To get expected behavior when creating a label verification or adjustment job, carefully verify your input data.

- If you are using image data, verify that your manifest file contains hexadecimal RGB color information.
- To save money on processing costs, filter your data to ensure you are not including unwanted objects in your labeling job input manifest.

- Add required Amazon S3 permissions to ensure your input data is processed correctly.

When you create an adjustment or verification labeling job using the Ground Truth API, you *must* use a different `LabelAttributeName` than the original labeling job.

## Color Information Requirements for Semantic Segmentation Jobs

To properly reproduce color information in verification or adjustment tasks, the tool requires hexadecimal RGB color information in the manifest (for example, #FFFFFF for white). When you set up a Semantic Segmentation verification or adjustment job, the tool examines the manifest to determine if this information is present. If it can't find it, Amazon SageMaker Ground Truth displays an error message and the ends job setup.

In prior iterations of the Semantic Segmentation tool, category color information wasn't output in hexadecimal RGB format to the output manifest. That feature was introduced to the output manifest at the same time the verification and adjustment workflows were introduced. Therefore, older output manifests aren't compatible with this new workflow.

## Filter Your Data Before Starting the Job

Amazon SageMaker Ground Truth processes all objects in your input manifest. If you have a partially labeled data set, you might want to create a custom manifest using an [Amazon S3 Select query](#) on your input manifest. Unlabeled objects individually fail, but they don't cause the job to fail, and they might incur processing costs. Filtering out objects you don't want verified reduces your costs.

If you create a verification job using the console, you can use the filtering tools provided there. If you create jobs using the API, make filtering your data part of your workflow where needed.

# Creating Custom Labeling Workflows

This document guides you through the process of setting up a workflow with a custom labeling template. For more information about starting a labeling job, see [Getting started \(p. 166\)](#). In that section, when you choose the **Task type**, select **Custom labeling task**, and then follow this section's instructions to configure it.

### Topics

- [Step 1: Setting up your workforce \(p. 300\)](#)
- [Step 2: Creating your custom worker task template \(p. 301\)](#)
- [Step 3: Processing with Amazon Lambda \(p. 307\)](#)
- [Demo Template: Annotation of Images with crowd-bounding-box \(p. 321\)](#)
- [Demo Template: Labeling Intents with crowd-classifier \(p. 325\)](#)
- [Custom Workflows via the API \(p. 332\)](#)

## Step 1: Setting up your workforce

In this step you use the console to establish which worker type to use and make the necessary sub-selections for the worker type. It assumes you have already completed the steps up to this point in the [Getting started \(p. 166\)](#) section and have chosen the **Custom labeling task** as the **Task type**.

### To configure your workforce.

1. First choose an option from the **Worker types**. There are three types currently available:

- **Public** uses an on-demand workforce of independent contractors, powered by Amazon Mechanical Turk. They are paid on a per-task basis.
  - **Private** uses your employees or contractors for handling data that needs to stay within your organization.
  - **Vendor** uses third party vendors that specialize in providing data labeling services, available via the Amazon Marketplace.
2. If you choose the **Public** option, you are asked to set the **number of workers per dataset object**. Having more than one worker perform the same task on the same object can help increase the accuracy of your results. The default is three. You can raise or lower that depending on the accuracy you need.

You are also asked to set a **price per task** by using a drop-down menu. The menu recommends price points based on how long it will take to complete the task.

The recommended method to determine this is to first run a short test of your task with a **private** workforce. The test provides a realistic estimate of how long the task takes to complete. You can then select the range your estimate falls within on the **Price per task** menu. If your average time is more than 5 minutes, consider breaking your task into smaller units.

## Next

[Step 2: Creating your custom worker task template \(p. 301\)](#)

## Step 2: Creating your custom worker task template

A **worker task template** is a file used by **Ground Truth** to customize the worker user interface (UI), or **human task UI**. You can create a worker task template using **HTML**, **CSS**, **JavaScript**, **Liquid template language**, and **Crowd HTML Elements**. **Liquid** is used to automate the template, and **Crowd HTML Elements** can be used to include common annotation tools and provide the logic to submit to **Ground Truth**.

- [Starting with a base template \(p. 301\)](#)
- [Developing templates locally \(p. 302\)](#)
- [Using External Assets \(p. 302\)](#)
- [Track your variables \(p. 302\)](#)
- [A simple sample \(p. 303\)](#)
- [Adding automation with Liquid \(p. 304\)](#)
- [End-to-end demos \(p. 306\)](#)
- [Next \(p. 307\)](#)

Use the following topics to learn how you can create a worker task template. You can see a repository of example **Ground Truth** worker task templates on [GitHub](#).

### Starting with a base template

You can use a template editor in the **Ground Truth** console to start creating a template. This editor includes a number of pre-designed base templates and an **HTML** and **Crowd HTML Element** autofill feature.

#### To access the **Ground Truth** custom template editor:

1. Following the instructions in [Create a Labeling Job \(Console\) \(p. 336\)](#) and select **Custom** for the labeling job **Task type**.

2. When you select **Next**, you will be able to access the template editor and base templates in the **Custom labeling task setup** section.
3. (Optional) Select a base template from the drop-down menu under **Templates**. If you prefer to create a template from scratch, choose **Custom** from the drop down-menu for a minimal template skeleton.

## Developing templates locally

While you need to be in the console to test how your template will process incoming data, you can test the look and feel of your template's HTML and custom elements in your browser by adding this code to the top of your HTML file.

### Example

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
```

This loads the necessary code to render the custom HTML elements. Use this if you want to develop your template's look and feel in your preferred editor rather than in the console.

Remember, though, this will not parse your variables. You may want to replace them with sample content while developing locally.

## Using External Assets

Amazon SageMaker Ground Truth custom templates allow external scripts and style sheets to be embedded. For example, the following code block demonstrates how you would add a style sheet located at <https://www.example.com/my-enhancement-styles.css> to your template.

### Example

```
<script src="https://www.example.com/my-enhancement-script.js"></script>
<link rel="stylesheet" type="text/css" href="https://www.example.com/my-enhancement-
styles.css">
```

If you encounter errors, ensure that your originating server is sending the correct MIME type and encoding headers with the assets.

For example, the MIME and encoding types for remote scripts are: `application/javascript;CHARSET=UTF-8`.

The MIME and encoding type for remote stylesheets are: `text/css;CHARSET=UTF-8`.

## Track your variables

In the process of building the sample below, there will be a step that adds variables to it to represent the pieces of data that may change from task to task, worker to worker. If you're starting with one of the sample templates, you will need to make sure you're aware of the variables it already uses. When you create your pre-annotation Amazon Lambda script, its output will need to contain values for any of those variables you choose to keep.

The values you use for the variables can come from your manifest file. All the key-value pairs in your data object are provided to your pre-annotation Lambda. If it's a simple pass-through script, matching

keys for values in your data object to variable names in your template is the easiest way to pass those values through to the tasks forms your workers see.

## A simple sample

All tasks begin and end with the `<crowd-form>` `</crowd-form>` elements. Like standard HTML `<form>` elements, all of your form code should go between them.

For a simple tweet-analysis task, use the `<crowd-classifier>` element. It requires the following attributes:

- *name* - the variable name to use for the result in the form output.
- *categories* - a JSON formatted array of the possible answers.
- *header* - a title for the annotation tool

As children of the `<crowd-classifier>` element, you must have three regions.

- `<classification-target>` - the text the worker will classify based on the options specified in the *categories* attribute above.
- `<full-instructions>` - instructions that are available from the "View full instructions" link in the tool. This can be left blank, but it is recommended that you give good instructions to get better results.
- `<short-instructions>` - a more brief description of the task that appears in the tool's sidebar. This can be left blank, but it is recommended that you give good instructions to get better results.

A simple version of this tool would look like this.

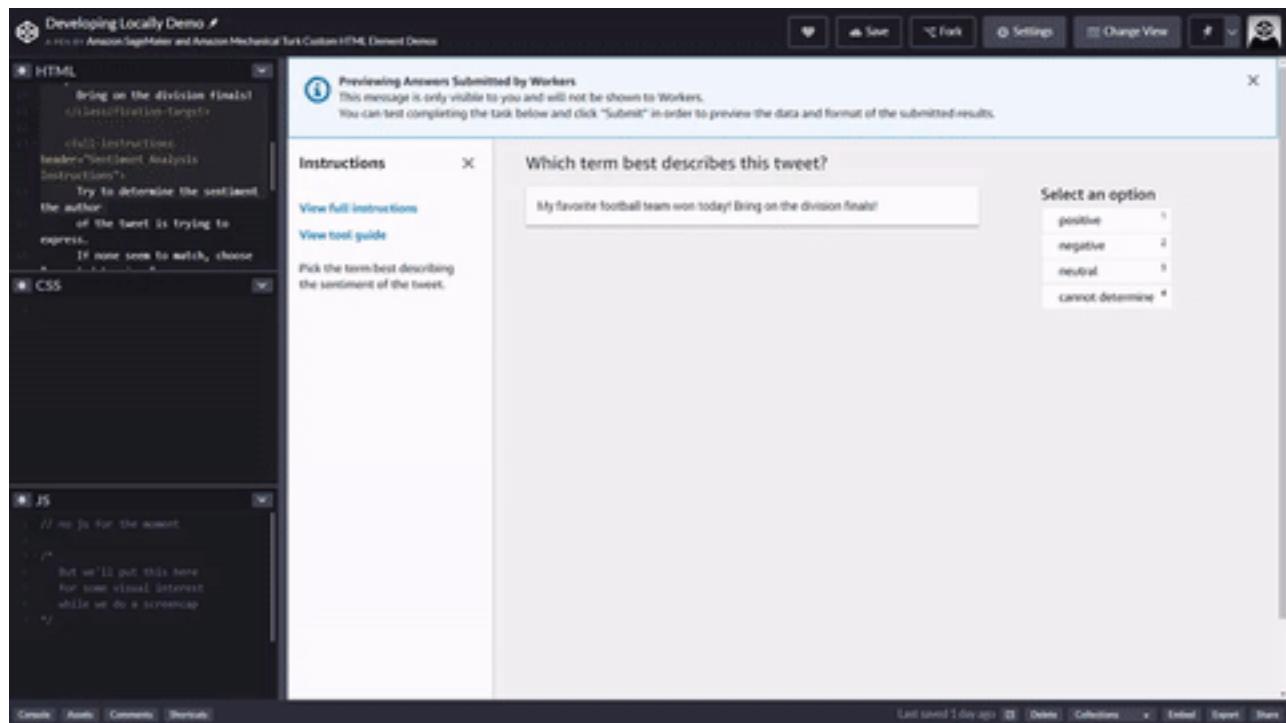
### Example of using `crowd-classifier`

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-classifier
    name="tweetFeeling"
    categories="['positive', 'negative', 'neutral', 'unclear']"
    header="Which term best describes this tweet?">
    <classification-target>
      My favorite football team won today!
      Bring on the division finals!
    </classification-target>

    <full-instructions header="Sentiment Analysis Instructions">
      Try to determine the sentiment the author
      of the tweet is trying to express.
      If none seem to match, choose "cannot determine."
    </full-instructions>

    <short-instructions>
      Pick the term best describing the sentiment
      of the tweet.
    </short-instructions>
  </crowd-classifier>
</crowd-form>
```

You can copy and paste the code into the editor in the Ground Truth labeling job creation workflow to preview the tool, or try out a [demo of this code on CodePen](#).



## Adding automation with Liquid

Our custom template system uses [Liquid](#) for automation. It is an open source inline markup language. In Liquid, the text between single curly braces and percent symbols is an instruction or *tag* that performs an operation like control flow or iteration. Text between double curly braces is a variable or *object* that outputs its value.

The most common use of Liquid will be to parse the data coming from your [pre-annotation Lambda](#) and pull out the relevant variables to create the task. The `taskInput` object returned by your [Pre-annotation Lambda \(p. 308\)](#) will be available as the `task.input` object in your templates.

The properties in your manifest's data objects are passed into your [Pre-annotation Lambda \(p. 308\)](#) as the `event.dataObject`. A simple pass-through script simply returns that object as the `taskInput` object. You would represent values from your manifest as variables as follows.

### Example Manifest data object

```
{
  "source": "This is a sample text for classification",
  "labels": [ "angry" , "sad" , "happy" , "inconclusive" ],
  "header": "What emotion is the speaker feeling?"
}
```

### Example Sample HTML using variables

```
<crowd-classifier
  name='tweetFeeling'
  categories='{{ task.input.labels | to_json }}'
  header='{{ task.input.header }}' >
<classification-target>
  {{ task.input.source }}
</classification-target>
```

Note the addition of " | to\_json" to the labels property above. That's a filter to turn the array into a JSON representation of the array. Variable filters are explained in the next section.

The following list includes two types of Liquid tags that you may find useful to automate template input data processing. If you select one of the following tag-types, you will be redirected to the Liquid documentation.

- **Control flow:** Includes programming logic operators like `if/else`, `unless`, and `case/when`.
- **Iteration:** Enables you to run blocks of code repeatedly using statements like `for` loops.

For an example of an HTML template that uses Liquid elements to create a `for` loop, see [translation-review-and-correction.liquid.html](#) in GitHub.

For more information and documentation, visit the [Liquid homepage](#).

### Variable filters

In addition to the standard [Liquid filters](#) and actions, Ground Truth offers a few additional filters. Filters are applied by placing a pipe (|) character after the variable name, then specifying a filter name. Filters can be chained in the form of:

#### Example

```
{ {<content>} | <filter> | <filter> }
```

#### Autoescape and explicit escape

By default, inputs will be HTML escaped to prevent confusion between your variable text and HTML. You can explicitly add the `escape` filter to make it more obvious to someone reading the source of your template that the escaping is being done.

#### [escape\\_once](#)

`escape_once` ensures that if you've already escaped your code, it doesn't get re-escaped on top of that. For example, so that `&` doesn't become `&&`;

#### [skip\\_autoescape](#)

`skip_autoescape` is useful when your content is meant to be used as HTML. For example, you might have a few paragraphs of text and some images in the full instructions for a bounding box.

#### **Use `skip_autoescape` sparingly**

The best practice in templates is to avoid passing in functional code or markup with `skip_autoescape` unless you are absolutely sure you have strict control over what's being passed. If you're passing user input, you could be opening your workers up to a Cross Site Scripting attack.

#### [to\\_json](#)

`to_json` will encode what you feed it to JSON (JavaScript Object Notation). If you feed it an object, it will serialize it.

#### [grant\\_read\\_access](#)

`grant_read_access` takes an S3 URI and encodes it into an HTTPS URL with a short-lived access token for that resource. This makes it possible to display to workers photo, audio, or video objects stored in S3 buckets that are not otherwise publicly accessible.

## Example of the filters

### Input

```
auto-escape: {{ "Have you read 'James & the Giant Peach'?" }}
explicit escape: {{ "Have you read 'James & the Giant Peach'?" | escape }}
explicit escape_once: {{ "Have you read 'James & the Giant Peach'?" | escape_once }}
skip_autoescape: {{ "Have you read 'James & the Giant Peach'?" | skip_autoescape }}
to_json: {{ jsObject | to_json }}
grant_read_access: {{ "s3://mybucket/myphoto.png" | grant_read_access }}
```

### Example

#### Output

```
auto-escape: Have you read 'James & the Giant Peach'?
explicit escape: Have you read 'James & the Giant Peach'?
explicit escape_once: Have you read 'James & the Giant Peach'?
skip_autoescape: Have you read 'James & the Giant Peach'?
to_json: { "point_number": 8, "coords": [ 59, 76 ] }
grant_read_access: https://s3.amazonaws.com/mybucket/myphoto.png?<access token and other
params>
```

## Example of an automated classification template.

To automate the simple text classification sample, replace the tweet text with a variable.

The text classification template is below with automation added. The changes/additions are highlighted in bold.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-classifier
    name="tweetFeeling"
    categories="['positive', 'negative', 'neutral', 'cannot determine']"
    header="Which term best describes this tweet?"
  >
    <classification-target>
      {{ task.input.source }}
    </classification-target>

    <full-instructions header="Analyzing a sentiment">
      Try to determine the feeling the author
      of the tweet is trying to express.
      If none seem to match, choose "other."
    </full-instructions>

    <short-instructions>
      Pick the term best describing the sentiment
      of the tweet.
    </short-instructions>
  </crowd-classifier>
</crowd-form>
```

The tweet text that was in the prior sample is now replaced with an object. The `entry.taskInput` object uses `source` (or another name you specify in your pre-annotation Lambda) as the property name for the text and it is inserted directly in the HTML by virtue of being between double curly braces.

## End-to-end demos

You can view the following end-to-end demos which include sample Lambda function:

- [Demo Template: Annotation of Images with crowd-bounding-box \(p. 321\)](#)
- [Demo Template: Labeling Intents with crowd-classifier \(p. 325\)](#)

## Next

[Step 3: Processing with Amazon Lambda \(p. 307\)](#)

## Step 3: Processing with Amazon Lambda

In this step, you learn how to create and specify the two types of [Amazon Lambda](#) functions that are required to create a custom labeling workflow:

- *Pre-annotation Lambda*: This function initiates for and pre-processes each data object sent to your labeling job prior to sending it to workers.
- *Post-annotation Lambda*: This function processes the results once workers submit a task. If you specify multiple workers per data object, this function may include logic to consolidate annotations.

If you are a new user of Lambda and Ground Truth, we recommend that you use the pages in this section as follows:

1. First, review [Pre-annotation and Post-annotation Lambda Function Requirements \(p. 307\)](#).
2. Then, use the page [Required Permissions To Use Amazon Lambda With Ground Truth \(p. 314\)](#) to learn about security and permission requirements to use your pre-annotation and post-annotation Lambda functions in a Ground Truth custom labeling job.
3. Next, you need to visit the Lambda console or use Lambda's APIs to create your functions. Use the section [Create Lambda Functions for a Custom Labeling Workflow \(p. 318\)](#) to learn how to create Lambda functions.
4. To learn how to test your Lambda functions, see [Test Pre-Annotation and Post-Annotation Lambda Functions \(p. 318\)](#).
5. After you create pre-processing and post-processing Lambda functions, select them from the **Lambda functions** section that comes after the code editor for your custom HTML in the Ground Truth console. To learn how to use these functions in a [CreateLabelingJob API request](#), see [Create a Labeling Job \(API\) \(p. 338\)](#).

For a custom labeling workflow tutorial that includes example pre-annotation and post-annotation Lambda functions, in the "[Demo Template: Annotation of Images with crowd-bounding-box \(p. 321\)](#)" document.

### Topics

- [Pre-annotation and Post-annotation Lambda Function Requirements \(p. 307\)](#)
- [Required Permissions To Use Amazon Lambda With Ground Truth \(p. 314\)](#)
- [Create Lambda Functions for a Custom Labeling Workflow \(p. 318\)](#)
- [Test Pre-Annotation and Post-Annotation Lambda Functions \(p. 318\)](#)

## Pre-annotation and Post-annotation Lambda Function Requirements

Use this section to learn about the syntax of the requests sent to pre-annotation and post-annotation Lambda functions, and the response syntax that Ground Truth requires to run a custom labeling workflow.

### Topics

- [Pre-annotation Lambda \(p. 308\)](#)
- [Post-annotation Lambda \(p. 310\)](#)

### Pre-annotation Lambda

Before a labeling task is sent to the worker, your pre-annotation Lambda function is invoked.

Ground Truth sends your Lambda function a JSON-formatted request to provide details about the labeling job and the data object. The following table contains the pre-annotation request schemas. Each parameter is described below.

Data object identified with "source-ref"

```
{
    "version": "2018-10-16",
    "labelingJobArn": <labelingJobArn>
    "dataObject" : {
        "source-ref": <s3Uri>
    }
}
```

Data object identified with "source"

```
{
    "version": "2018-10-16",
    "labelingJobArn": <labelingJobArn>
    "dataObject" : {
        "source": <string>
    }
}
```

- **version (string):** This is a version number used internally by Ground Truth.
- **labelingJobArn (string):** This is the Amazon Resource Name, or ARN, of your labeling job. This ARN can be used to reference the labeling job when using Ground Truth API operations such as `DescribeLabelingJob`.
- The **dataObject (JSON object):** The key contains a single JSON line, either from your input manifest file or sent from Amazon SNS. The JSON line objects in your manifest can be up to 100 kilobytes in size and contain a variety of data. For a very basic image annotation job, the `dataObject` JSON may just contain a `source-ref` key, identifying the image to be annotated. If the data object (for example, a line of text) is included directly in the input manifest file, the `data object` is identified with `source`. If you create a verification or adjustment job, this line may contain label data and metadata from the previous labeling job.

The following table includes code block examples of a pre-annotation request. Each parameter in these example requests is explained below the tabbed table.

Data object identified with "source-ref"

```
{
    "version": "2018-10-16",
    "labelingJobArn": "arn:aws:sagemaker:<aws_region>:<aws_account_number>:labeling-
job/<labeling_job_name>"
    "dataObject" : {
        "source-ref": "s3://<input-data-bucket>/<data-object-file-name>"
    }
}
```

Data object identified with "source"

```
{
    "version": "2018-10-16",
    "labelingJobArn": "arn:aws:sagemaker:<aws_region>:<aws_account_number>:labeling-
job/<labeling_job_name>"
    "dataObject" : {
        "source": "Sue purchased 10 shares of the stock on April 10th, 2020"
    }
}
```

In return, Ground Truth requires a response formatted like the following:

#### Example of expected return data

```
{
    "taskInput": <json object>,
    "isHumanAnnotationRequired": <boolean> # Optional
}
```

In the previous example, the `<json object>` needs to contain *all* the data your custom worker task template needs. If you're doing a bounding box task where the instructions stay the same all the time, it may just be the HTTP(S) or Amazon S3 resource for your image file. If it's a sentiment analysis task and different objects may have different choices, it is the object reference as a string and the choices as an array of strings.

#### Implications of `isHumanAnnotationRequired`

This value is optional because it defaults to `true`. The primary use case for explicitly setting it is when you want to exclude this data object from being labeled by human workers.

If you have a mix of objects in your manifest, with some requiring human annotation and some not needing it, you can include a `isHumanAnnotationRequired` value in each data object. You can add logic to your pre-annotation Lambda to dynamically determine if an object requires annotation, and set this boolean value accordingly.

#### Examples of Pre-annotation Lambda Functions

The following, basic pre-annotation Lambda function accesses the JSON object in `dataObject` from the initial request, and returns it in the `taskInput` parameter.

```
import json

def lambda_handler(event, context):
    return {
        "taskInput": event['dataObject']
    }
```

Assuming the input manifest file uses "source-ref" to identify data objects, the worker task template used in the same labeling job as this pre-annotation Lambda must include a Liquid element like the following to ingest `dataObject`:

```
{{ task.input.source-ref | grant_read_access }}
```

If the input manifest file used `source` to identify the data object, the work task template can ingest `dataObject` with the following:

```
{{ task.input.source }}
```

The following pre-annotation Lambda example includes logic to identify the key used in `dataObject`, and to point to that data object using `taskObject` in the Lambda's return statement.

```

import json

def lambda_handler(event, context):

    # Event received
    print("Received event: " + json.dumps(event, indent=2))

    # Get source if specified
    source = event['dataObject']['source'] if "source" in event['dataObject'] else None

    # Get source-ref if specified
    source_ref = event['dataObject']['source-ref'] if "source-ref" in event['dataObject'] else None

    # if source field present, take that otherwise take source-ref
    task_object = source if source is not None else source_ref

    # Build response object
    output = {
        "taskInput": {
            "taskObject": task_object
        },
        "humanAnnotationRequired": "true"
    }

    print(output)
    # If neither source nor source-ref specified, mark the annotation failed
    if task_object is None:
        print(" Failed to pre-process {} !".format(event["labelingJobArn"]))
        output["humanAnnotationRequired"] = "false"

    return output

```

## Post-annotation Lambda

When all workers have annotated the data object or when `TaskAvailabilityLifetimeInSeconds` has been reached, whichever comes first, Ground Truth sends those annotations to your post-annotation Lambda. This Lambda is generally used for [Consolidate Annotations \(p. 429\)](#).

### Tip

To see an example of a post-consolidation Lambda function, see [annotation\\_consolidation\\_lambda.py](#) in the [aws-sagemaker-ground-truth-recipe](#) GitHub repository.

The following code block contains the post-annotation request schema. Each parameter is described in the following bulleted list.

```
{
    "version": "2018-10-16",
    "labelingJobArn": <string>,
    "labelCategories": [<string>],
    "labelAttributeName": <string>,
    "roleArn" : <string>,
    "payload": {
        "s3Uri": <string>
    }
}
```

- `version (string)`: A version number used internally by Ground Truth.

- **labelingJobArn (string)**: The Amazon Resource Name, or ARN, of your labeling job. This ARN can be used to reference the labeling job when using Ground Truth API operations such as `DescribeLabelingJob`.
- **labelCategories (list of strings)**: Includes the label categories and other attributes you either specified in the console, or that you include in the label category configuration file.
- **labelAttributeName (string)**: Either the name of your labeling job, or the label attribute name you specify when you create the labeling job.
- **roleArn (string)**: The Amazon Resource Name (ARN) of the IAM execution role you specify when you create the labeling job.
- **payload (JSON object)**: A JSON that includes an `s3Uri` key, which identifies the location of the annotation data for that data object in Amazon S3. The second code block below shows an example of this annotation file.

The following code block contains an example of a post-annotation request. Each parameter in this example request is explained below the code block.

#### **Example of an post-annotation Lambda request**

```
{
    "version": "2018-10-16",
    "labelingJobArn": "arn:aws:sagemaker:us-west-2:111122223333:labeling-job/labeling-job-name",
    "labelCategories": ["Ex Category1", "Ex Category2", "Ex Category3"],
    "labelAttributeName": "labeling-job-attribute-name",
    "roleArn" : "arn:aws:iam::111122223333:role/role-name",
    "payload": {
        "s3Uri": "s3://annotations.json"
    }
}
```

#### **Note**

If no worker works on the data object and `TaskAvailabilityLifetimeInSeconds` has been reached, the data object is marked as failed and not included as part of post-annotation Lambda invocation.

The following code block contains the payload schema. This is the file that is indicated by the `s3Uri` parameter in the post-annotation Lambda request payload JSON object. For example, if the previous code block is the post-annotation Lambda request, the following annotation file is located at `s3://annotations.json`.

Each parameter is described in the following bulleted list.

#### **Example of an annotation file**

```
[
    {
        "datasetObjectId": <string>,
        "dataObject": {
            "s3Uri": <string>,
            "content": <string>
        },
        "annotations": [
            {
                "workerId": <string>,
                "annotationData": {
                    "content": <string>,
                    "s3Uri": <string>
                }
            }
        ]
    }
]
```

```
        }]
    }
]
```

- **datasetObjectId** (string): Identifies a unique ID that Ground Truth assigns to each data object you send to the labeling job.
- **dataObject** (JSON object): The data object that was labeled. If the data object is included in the input manifest file and identified using the `source` key (for example, a string), `dataObject` includes a `content` key, which identifies the data object. Otherwise, the location of the data object (for example, a link or S3 URI) is identified with `s3Uri`.
- **annotations** (list of JSON objects): This list contains a single JSON object for each annotation submitted by workers for that `dataObject`. A single JSON object contains a unique `workerId` that can be used to identify the worker that submitted that annotation. The `annotationData` key contains one of the following:
  - **content** (string): Contains the annotation data.
  - **s3Uri** (string): Contains an S3 URI that identifies the location of the annotation data.

The following table contains examples of the content that you may find in payload for different types of annotation.

#### Named Entity Recognition Payload

```
[ {
  {
    "datasetObjectId": "1",
    "dataObject": {
      "content": "Sift 3 cups of flour into the bowl."
    },
    "annotations": [
      {
        "workerId": "private.us-west-2.ef7294f850a3d9d1",
        "annotationData": {
          "content": "{\"crowd-entity-annotation\":{\"entities\":[{\\"endOffset\":4,\\\"label\":\"verb\",\\\"startOffset\":0},{\\\"endOffset\":6,\\\"label\":\"number\\\",\\\"startOffset\":5},{\\\"endOffset\":20,\\\"label\":\"object\",\\\"startOffset\":15},{\\\"endOffset\":34,\\\"label\":\"object\",\\\"startOffset\":30}]}"
        }
      }
    ]
  }
}
```

#### Semantic Segmentation Payload

```
[ {
  {
    "datasetObjectId": "2",
    "dataObject": {
      "s3Uri": "s3://gt-input-data/images/bird3.jpg"
    },
    "annotations": [
      {
        "workerId": "private.us-west-2.ab1234c5678a919d0",
        "annotationData": {
          "content": "{\"crowd-segmentation\":{\"inputImageProperties\":{\"height\":2000,\\\"width\":3020},\\\"labelMappings\":[\\\"Bird\\\":{\\\"color\\\":\\\"#2ca02c\\\"}}},\\\"labeledImage\\\":{\\\"pngImageData\\\":\\\"iVBOR...\\\"}}}"
        }
      }
    ]
  }
}
```

```
        ]
    }
]
```

### Bounding Box Payload

```
[
  {
    "datasetObjectId": "0",
    "dataObject": {
      "s3Uri": "s3://gt-input-data/images/bird1.jpg"
    },
    "annotations": [
      {
        "workerId": "private.us-west-2.ab1234c5678a919d0",
        "annotationData": {
          "content": "{\"boundingBox\":{\"boundingBoxes\":[{\\"height\":2052,\\"label\\":\"Bird\",\\\"left\":583,\\"top\":302,\\"width\":1375}],\\\"inputImageProperties\\\":{\\"height\":2497,\\"width\":3745}}}"
        }
      }
    ]
  ]
]
```

Your post-annotation Lambda function may contain logic similar to the following to loop through and access all annotations contained in the request. For a full example, see [annotation\\_consolidation\\_lambda.py](#) in the [aws-sagemaker-ground-truth-recipe](#) GitHub repository. In this GitHub example, you must add your own annotation consolidation logic.

```
for i in range(len(annotations)):
    worker_id = annotations[i]["workerId"]
    annotation_content = annotations[i]['annotationData'].get('content')
    annotation_s3_uri = annotations[i]['annotationData'].get('s3uri')
    annotation = annotation_content if annotation_s3_uri is None else
        s3_client.get_object_from_s3(
            annotation_s3_uri)
    annotation_from_single_worker = json.loads(annotation)

    print("{} Received Annotations from worker [{}] is [{}]"
          .format(log_prefix, worker_id, annotation_from_single_worker))
```

### Tip

When you run consolidation algorithms on the data, you can use an Amazon database service to store results, or you can pass the processed results back to Ground Truth. The data you return to Ground Truth is stored in consolidated annotation manifests in the S3 bucket specified for output during the configuration of the labeling job.

In return, Ground Truth requires a response formatted like the following:

### Example of expected return data

```
[
  {
    "datasetObjectId": <string>,
    "consolidatedAnnotation": {
      "content": {
        "<labelattributename>": {
          "# ... label content
        }
      }
    }
]
```

```

        }
    },
{
    "datasetObjectId": <string>,
    "consolidatedAnnotation": {
        "content": {
            "<labelattributename>": {
                "# ... label content
            }
        }
    }
}.
.
.
]
```

At this point, all the data you're sending to your S3 bucket, other than the datasetObjectId, is in the content object.

When you return annotations in content, this results in an entry in your job's output manifest like the following:

#### Example of label format in output manifest

```
{
    "source-ref"/"source" : "<s3uri or content>",
    "<labelAttributeName>": {
        "# ... label content from you
    },
    "<labelAttributeName>-metadata": { # This will be added by Ground Truth
        "job_name": <labelingJobName>,
        "type": "groundTruth/custom",
        "human-annotated": "yes",
        "creation_date": <date> # Timestamp of when received from Post-labeling Lambda
    }
}
```

Because of the potentially complex nature of a custom template and the data it collects, Ground Truth does not offer further processing of the data.

### Required Permissions To Use Amazon Lambda With Ground Truth

You may need to configure some or all the following to create and use Amazon Lambda with Ground Truth.

- You need to grant an IAM role or user (collectively, an IAM entity) permission to create the pre-annotation and post-annotation Lambda functions using Amazon Lambda, and to choose them when creating the labeling job.
- The IAM execution role specified when the labeling job is configured needs permission to invoke the pre-annotation and post-annotation Lambda functions.
- The post-annotation Lambda functions may need permission to access Amazon S3.

Use the following sections to learn how to create the IAM entities and grant permissions described above.

#### Topics

- [Grant Permission to Create and Select an Amazon Lambda Function \(p. 315\)](#)

- [Grant IAM Execution Role Permission to Invoke Amazon Lambda Functions \(p. 315\)](#)
- [Grant Post-Annotation Lambda Permissions to Access Annotation \(p. 316\)](#)

### Grant Permission to Create and Select an Amazon Lambda Function

If you do not require granular permissions to develop pre-annotation and post-annotation Lambda functions, you can attach the Amazon managed policy `AWSLambda_FullAccess` to an IAM user or role. This policy grants broad permissions to use all Lambda features, as well as permission to perform actions in other Amazon services with which Lambda interacts.

To create a more granular policy for security-sensitive use cases, refer to the documentation [Identity-based IAM policies for Lambda](#) in the [Amazon Lambda Developer Guide](#) to learn how to create an IAM policy that fits your use case.

### Policies to Use the Lambda Console

If you want to grant an IAM entity permission to use the Lambda console, see [Using the Lambda console](#) in the [Amazon Lambda Developer Guide](#).

Additionally, if you want the user to be able to access and deploy the Ground Truth starter pre-annotation and post-annotation functions using the Amazon Serverless Application Repository in the Lambda console, you must specify the `<aws-region>` where you want to deploy the functions (this should be the same Amazon Region used to create the labeling job), and add the following policy to the IAM role.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "VisualEditor0",  
            "Effect": "Allow",  
            "Action": [  
                "serverlessrepo>ListApplicationVersions",  
                "serverlessrepo>GetApplication",  
                "serverlessrepo>CreateCloudFormationTemplate"  
            ],  
            "Resource": "arn:aws:serverlessrepo:<aws-region>:838997950401:applications/aws-sagemaker-ground-truth-recipe"  
        },  
        {  
            "Sid": "VisualEditor1",  
            "Effect": "Allow",  
            "Action": "serverlessrepo/SearchApplications",  
            "Resource": "*"  
        }  
    ]  
}
```

### Policies to See Lambda Functions in the Ground Truth Console

To grant an IAM entity permission to view Lambda functions in the Ground Truth console when the user is creating a custom labeling job, the entity must have the permissions described in [Grant IAM Permission to Use the Amazon SageMaker Ground Truth Console \(p. 441\)](#), including the permissions described in the section [Custom Labeling Workflow Permissions \(p. 444\)](#).

### Grant IAM Execution Role Permission to Invoke Amazon Lambda Functions

If you add the IAM managed policy `AmazonSageMakerGroundTruthExecution` to the IAM execution role used to create the labeling job, this role has permission to list and invoke Lambda functions with one

of the following strings in the function name: `GtRecipe`, `SageMaker`, `Sagemaker`, `sagemaker`, or `LabelingFunction`.

If the pre-annotation or post-annotation Lambda function names do not include one of the terms in the preceding paragraph, or if you require more granular permission than those in the `AmazonSageMakerGroundTruthExecution` managed policy, you can add a policy similar to the following to give the execution role permission to invoke pre-annotation and post-annotation functions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action":
                "lambda:InvokeFunction",
            "Resource": [
                "arn:aws:lambda:<region>:<account-id>:function:<pre-annotation-lambda-name>",
                "arn:aws:lambda:<region>:<account-id>:function:<post-annotation-lambda-name>"
            ]
        }
    ]
}
```

### Grant Post-Annotation Lambda Permissions to Access Annotation

As described in [Post-annotation Lambda \(p. 310\)](#), the post-annotation Lambda request includes the location of the annotation data in Amazon S3. This location is identified by the `s3Uri` string in the payload object. To process the annotations as they come in, even for a simple pass through function, you need to assign the necessary permissions to the post-annotation [Lambda execution role](#) to read files from the Amazon S3.

There are many ways that you can configure your Lambda to access annotation data in Amazon S3. Two common ways are:

- Allow the Lambda execution role to assume the SageMaker execution role identified in `roleArn` in the post-annotation Lambda request. This SageMaker execution role is the one used to create the labeling job, and has access to the Amazon S3 output bucket where the annotation data is stored.
- Grant the Lambda execution role permission to access the Amazon S3 output bucket directly.

Use the following sections to learn how to configure these options.

#### Grant Lambda Permission to Assume SageMaker Execution Role

To allow a Lambda function to assume a SageMaker execution role, you must attach a policy to the Lambda function's execution role, and modify the trust relationship of the SageMaker execution role to allow Lambda to assume it.

1. [Attach the following IAM policy](#) to your Lambda function's execution role to assume the SageMaker execution role identified in `Resource`. Replace `222222222222` with an [Amazon account ID](#). Replace `sm-execution-role` with the name of the assumed role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::222222222222:role/sm-execution-role"
        }
    ]
}
```

```

        "Resource": "arn:aws:iam::222222222222:role/sm-execution-role"
    }
}
```

2. **Modify the trust policy** of the SageMaker execution role to include the following Statement. Replace **222222222222** with an [Amazon account ID](#). Replace **my-lambda-execution-role** with the name of the assumed role.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::222222222222:role/my-lambda-execution-role"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

### Grant Lambda Execution Role Permission to Access S3

You can add a policy similar to the following to the post-annotation Lambda function execution role to give it S3 read permissions. Replace  with the name of the output bucket you specify when you create a labeling job.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::/*"
        }
    ]
}
```

To add S3 read permissions to a Lambda execution role in the Lambda console, use the following procedure.

#### Add S3 read permissions to post-annotation Lambda:

1. Open the [Functions page](#) in the Lambda console.
2. Choose the name of the post-annotation function.
3. Choose **Configuration** and then choose **Permissions**.
4. Select the **Role name** and the summary page for that role opens in the IAM console in a new tab.
5. Select **Attach policies**.
6. Do one of the following:
  - Search for and select **AmazonS3ReadOnlyAccess** to give the function permission to read all buckets and objects in the account.
  - If you require more granular permissions, select **Create policy** and use the policy example in the preceding section to create a policy. Note that you must navigate back to the execution role summary page after you create the policy.

7. If you used the `AmazonS3ReadOnlyAccess` managed policy, select **Attach policy**.

If you created a new policy, navigate back to the Lambda execution role summary page and attach the policy you just created.

## Create Lambda Functions for a Custom Labeling Workflow

You can create a Lambda function using the Lambda console, the Amazon CLI, or an Amazon SDK in a supported programming language of your choice. Use the Amazon Lambda Developer Guide to learn more about each of these options:

- To learn how to create a Lambda function using the console, see [Create a Lambda function with the console](#).
- To learn how to create a Lambda function using the Amazon CLI, see [Using Amazon Lambda with the Amazon Command Line Interface](#).
- Select the relevant section in the table of contents to learn more about working with Lambda in the language of your choice. For example, select [Working with Python](#) to learn more about using Lambda with the Amazon SDK for Python (`Boto3`).

Ground Truth provides pre-annotation and post-annotation templates through an Amazon Serverless Application Repository (SAR) *recipe*. Use the following procedure to select the Ground Truth recipe in the Lambda console.

### Use the Ground Truth SAR recipe to create pre-annotation and post-annotation Lambda functions:

1. Open the [Functions page](#) on the Lambda console.
2. Select **Create function**.
3. Select **Browse serverless app repository**.
4. In the search text box, enter `aws-sagemaker-ground-truth-recipe` and select that app.
5. Select **Deploy**. The app may take a couple of minutes to deploy.

Once the app deploys, two functions appear in the **Functions** section of the Lambda console: `serverlessrepo-aws-sagema-GtRecipePreHumanTaskFunc-<id>` and `serverlessrepo-aws-sagema-GtRecipeAnnotationConsol-<id>`.

6. Select one of these functions and add your custom logic in the **Code** section.
7. When you are finished making changes, select **Deploy** to deploy them.

## Test Pre-Annotation and Post-Annotation Lambda Functions

You can test your pre-annotation and post annotation Lambda functions in the Lambda console. If you are a new user of Lambda, you can learn how to test, or *invoke*, your Lambda functions in the console using the [Create a Lambda function](#) tutorial with the console in the Amazon Lambda Developer Guide.

You can use the sections on this page to learn how to test the Ground Truth pre-annotation and post-annotation templates provided through an Amazon Serverless Application Repository (SAR).

### Topics

- [Prerequisites \(p. 319\)](#)
- [Test the Pre-annotation Lambda Function \(p. 319\)](#)
- [Test the Post-Annotation Lambda Function \(p. 320\)](#)

## Prerequisites

You must do the following to use the tests described on this page.

- You need access to the Lambda console, and you need permission to create and invoke Lambda functions. To learn how to set up these permissions, see [Grant Permission to Create and Select an Amazon Lambda Function \(p. 315\)](#).
- If you have not deployed the Ground Truth SAR recipe, use the procedure in [Create Lambda Functions for a Custom Labeling Workflow \(p. 318\)](#) to do so.
- To test the post-annotation Lambda function, you must have a data file in Amazon S3 with sample annotation data. For a simple test, you can copy and paste the following code into a file and save it as `sample-annotations.json` and [upload this file to Amazon S3](#). Note the S3 URI of this file—you need this information to configure the post-annotation Lambda test.

```
[{"datasetObjectId": "0", "dataObject": {"content": "To train a machine learning model, you need a large, high-quality, labeled dataset. Ground Truth helps you build high-quality training datasets for your machine learning models."}, "annotations": [{"workerId": "private.us-west-2.0123456789", "annotationData": {"content": "{\"crowd-entity-annotation\": {\"entities\": [{\"endOffset\": 8, \"label\": \"verb\", \"startOffset\": 3}, {\"endOffset\": 27, \"label\": \"adjective\", \"startOffset\": 11}, {\"endOffset\": 33, \"label\": \"object\", \"startOffset\": 28}, {\"endOffset\": 51, \"label\": \"adjective\", \"startOffset\": 46}, {\"endOffset\": 65, \"label\": \"adjective\", \"startOffset\": 53}, {\"endOffset\": 74, \"label\": \"adjective\", \"startOffset\": 67}, {\"endOffset\": 82, \"label\": \"adjective\", \"startOffset\": 75}, {\"endOffset\": 102, \"label\": \"verb\", \"startOffset\": 97}, {\"endOffset\": 112, \"label\": \"verb\", \"startOffset\": 107}, {\"endOffset\": 125, \"label\": \"adjective\", \"startOffset\": 113}, {\"endOffset\": 134, \"label\": \"adjective\", \"startOffset\": 126}, {\"endOffset\": 143, \"label\": \"object\", \"startOffset\": 135}, {\"endOffset\": 169, \"label\": \"adjective\", \"startOffset\": 153}, {\"endOffset\": 176, \"label\": \"object\", \"startOffset\": 170}]}]}}], {"datasetObjectId": "1", "dataObject": {"content": "Sift 3 cups of flour into the bowl."}, "annotations": [{"workerId": "private.us-west-2.0123456789", "annotationData": {"content": "{\"crowd-entity-annotation\": {\"entities\": [{\"endOffset\": 4, \"label\": \"verb\", \"startOffset\": 0}, {\"endOffset\": 6, \"label\": \"number\", \"startOffset\": 5}, {\"endOffset\": 20, \"label\": \"object\", \"startOffset\": 15}, {\"endOffset\": 34, \"label\": \"object\", \"startOffset\": 30}]}]}}], {"datasetObjectId": "2", "dataObject": {"content": "Jen purchased 10 shares of the stock on January 1st, 2020."}, "annotations": [{"workerId": "private.us-west-2.0123456789", "annotationData": {"content": "{\"crowd-entity-annotation\": {\"entities\": [{\"endOffset\": 3, \"label\": \"person\", \"startOffset\": 0}, {\"endOffset\": 13, \"label\": \"verb\", \"startOffset\": 4}, {\"endOffset\": 16, \"label\": \"number\", \"startOffset\": 14}, {\"endOffset\": 58, \"label\": \"date\", \"startOffset\": 40}]}]}}], {"datasetObjectId": "3", "dataObject": {"content": "The narrative was interesting, however the character development was weak."}, "annotations": [{"workerId": "private.us-west-2.0123456789", "annotationData": {"content": "{\"crowd-entity-annotation\": {\"entities\": [{\"endOffset\": 29, \"label\": \"adjective\", \"startOffset\": 18}, {\"endOffset\": 73, \"label\": \"adjective\", \"startOffset\": 69}]}]}}]
```

- You must use the directions in [Grant Post-Annotation Lambda Permissions to Access Annotation \(p. 316\)](#) to give your post-annotation Lambda function's execution role permission to assume the SageMaker execution role you use to create the labeling job. The post-annotation Lambda function uses the SageMaker execution role to access the annotation data file, `sample-annotations.json`, in S3.

## Test the Pre-annotation Lambda Function

Use the following procedure to test the pre-annotation Lambda function created when you deployed the Ground Truth Amazon Serverless Application Repository (SAR) recipe.

### Test the Ground Truth SAR recipe pre-annotation Lambda function

1. Open the [Functions page](#) in the Lambda console.

2. Select the pre-annotation function that was deployed from the Ground Truth SAR recipe. The name of this function is similar to `serverlessrepo-aws-sagema-GtRecipePreHumanTaskFunc-<id>`.
3. In the **Code source** section, select the arrow next to **Test**.
4. Select **Configure test event**.
5. Keep the **Create new test event** option selected.
6. Under **Event template**, select **SageMaker Ground Truth PreHumanTask**.
7. Give your test an **Event name**.
8. Select **Create**.
9. Select the arrow next to **Test** again and you should see that the test you created is selected, which is indicated with a dot by the event name. If it is not selected, select it.
10. Select **Test** to run the test.

After you run the test, you can see the **Execution results**. In the **Function logs**, you should see a response similar to the following:

```
START RequestId: cd117d38-8365-4e1a-bffb-0dc631a878f Version: $LATEST
Received event: {
  "version": "2018-10-16",
  "labelingJobArn": "arn:aws:sagemaker:us-east-2:123456789012:labeling-job/example-job",
  "dataObject": {
    "source-ref": "s3://sagemakerexample/object_to_annotate.jpg"
  }
}
{'taskInput': {'taskObject': 's3://sagemakerexample/object_to_annotate.jpg'},
 'isHumanAnnotationRequired': 'true'}
END RequestId: cd117d38-8365-4e1a-bffb-0dc631a878f
REPORT RequestId: cd117d38-8365-4e1a-bffb-0dc631a878f Duration: 0.42 ms Billed Duration: 1 ms Memory Size: 128 MB Max Memory Used: 43 MB
```

In this response, we can see the Lambda function's output matches the required pre-annotation response syntax:

```
{'taskInput': {'taskObject': 's3://sagemakerexample/object_to_annotate.jpg'},
 'isHumanAnnotationRequired': 'true'}
```

## Test the Post-Annotation Lambda Function

Use the following procedure to test the post-annotation Lambda function created when you deployed the Ground Truth Amazon Serverless Application Repository (SAR) recipe.

### Test the Ground Truth SAR recipe post-annotation Lambda

1. Open the [Functions page](#) in the Lambda console.
2. Select the post-annotation function that was deployed from the Ground Truth SAR recipe. The name of this function is similar to `serverlessrepo-aws-sagema-GtRecipeAnnotationConsol-<id>`.
3. In the **Code source** section, select the arrow next to **Test**.
4. Select **Configure test event**.
5. Keep the **Create new test event** option selected.
6. Under **Event template**, select **SageMaker Ground Truth AnnotationConsolidation**.
7. Give your test an **Event name**.
8. Modify the template code provided as follows:

- Replace the Amazon Resource Name (ARN) in `roleArn` with the ARN of the SageMaker execution role you used to create the labeling job.
- Replace the S3 URI in `s3Uri` with the URI of the `sample-annotations.json` file you added to Amazon S3.

After you make these modifications, your test should look similar to the following:

```
{  
    "version": "2018-10-16",  
    "labelingJobArn": "arn:aws:sagemaker:us-east-2:123456789012:labeling-job/example-  
job",  
    "labelAttributeName": "example-attribute",  
    "roleArn": "arn:aws:iam::222222222222:role/sm-execution-role",  
    "payload": {  
        "s3Uri": "s3://your-bucket/sample-annotations.json"  
    }  
}
```

9. Select **Create**.
10. Select the arrow next to **Test** again and you should see that the test you created is selected, which is indicated with a dot by the event name. If it is not selected, select it.
11. Select the **Test** to run the test.

After you run the test, you should see a -- Consolidated Output -- section in the **Function Logs**, which contains a list of all annotations included in `sample-annotations.json`.

## Demo Template: Annotation of Images with crowd-bounding-box

When you chose to use a custom template as your task type in the Amazon SageMaker Ground Truth console, you reach the **Custom labeling task panel**. There you can choose from multiple base templates. The templates represent some of the most common tasks and provide a sample to work from as you create your customized labeling task's template. If you are not using the console, or as an additional recourse, see [Amazon SageMaker Ground Truth Sample Task UIs](#) for a repository of demo templates for a variety of labeling job task types.

This demonstration works with the **BoundingBox** template. The demonstration also works with the Amazon Lambda functions needed for processing your data before and after the task. In the Github repository above, to find templates that work with Amazon Lambda functions, look for `{} task.input.<property name> {}` in the template.

### Topics

- [Starter Bounding Box custom template \(p. 321\)](#)
- [Your own Bounding Box custom template \(p. 322\)](#)
- [Your manifest file \(p. 323\)](#)
- [Your pre-annotation Lambda function \(p. 324\)](#)
- [Your post-annotation Lambda function \(p. 324\)](#)
- [The output of your labeling job \(p. 325\)](#)

### [Starter Bounding Box custom template](#)

This is the starter bounding box template that is provided.

```

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-bounding-box
    name="boundingBox"
    src="{{ task.input.taskObject | grant_read_access }}"
    header="{{ task.input.header }}"
    labels="{{ task.input.labels | to_json | escape }}"
  >

    <!-- The <full-instructions> tag is where you will define the full instructions of your
    task. -->
    <full-instructions header="Bounding Box Instructions" >
      <p>Use the bounding box tool to draw boxes around the requested target of interest:</p>
      <ol>
        <li>Draw a rectangle using your mouse over each instance of the target.</li>
        <li>Make sure the box does not cut into the target, leave a 2 - 3 pixel margin</li>
        <li>
          When targets are overlapping, draw a box around each object,
          include all contiguous parts of the target in the box.
          Do not include parts that are completely overlapped by another object.
        </li>
        <li>
          Do not include parts of the target that cannot be seen,
          even though you think you can interpolate the whole shape of the target.
        </li>
        <li>Avoid shadows, they're not considered as a part of the target.</li>
        <li>If the target goes off the screen, label up to the edge of the image.</li>
      </ol>
    </full-instructions>

    <!-- The <short-instructions> tag allows you to specify instructions that are displayed
    in the left hand side of the task interface.
    It is a best practice to provide good and bad examples in this section for quick
    reference. -->
    <short-instructions>
      Use the bounding box tool to draw boxes around the requested target of interest.
    </short-instructions>
  </crowd-bounding-box>
</crowd-form>

```

The custom templates use the [Liquid template language](#), and each of the items between double curly braces is a variable. The pre-annotation Amazon Lambda function should provide an object named `taskInput` and that object's properties can be accessed as `{{ task.input.<property name> }}` in your template.

## Your own Bounding Box custom template

As an example, assume you have a large collection of animal photos in which you know the kind of animal in an image from a prior image-classification job. Now you want to have a bounding box drawn around it.

In the starter sample, there are three variables: `taskObject`, `header`, and `labels`.

Each of these would be represented in different parts of the bounding box.

- `taskObject` is an HTTP(S) URL or S3 URI for the photo to be annotated. The added `| grant_read_access` is a filter that will convert an S3 URI to an HTTPS URL with short-lived access to that resource. If you're using an HTTP(S) URL, it's not needed.
- `header` is the text above the photo to be labeled, something like "Draw a box around the bird in the photo."

- `labels` is an array, represented as `[ 'item1', 'item2', ... ]`. These are labels that can be assigned by the worker to the different boxes they draw. You can have one or many.

Each of the variable names come from the JSON object in the response from your pre-annotation Lambda. The names above are merely suggested. Use whatever variable names make sense to you and will promote code readability among your team.

#### **Only use variables when necessary**

If a field will not change, you can remove that variable from the template and replace it with that text, otherwise you have to repeat that text as a value in each object in your manifest or code it into your pre-annotation Lambda function.

#### **Example : Final Customized Bounding Box Template**

To keep things simple, this template will have one variable, one label, and very basic instructions. Assuming your manifest has an "animal" property in each data object, that value can be re-used in two parts of the template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-bounding-box
    name="boundingBox"
    labels="[ '{{ task.input.animal }}' ]"
    src="{{ task.input.source-ref | grant_read_access }}"
    header="Draw a box around the {{ task.input.animal }}."
  >
    <full-instructions header="Bounding Box Instructions" >
      <p>Draw a bounding box around the {{ task.input.animal }} in the image. If there is more than one {{ task.input.animal }} per image, draw a bounding box around the largest one.</p>
      <p>The box should be tight around the {{ task.input.animal }} with no more than a couple of pixels of buffer around the edges.</p>
      <p>If the image does not contain a {{ task.input.animal }}, check the <strong> Nothing to label</strong> box.</p>
    </full-instructions>
    <short-instructions>
      <p>Draw a bounding box around the {{ task.input.animal }} in each image. If there is more than one {{ task.input.animal }} per image, draw a bounding box around the largest one.</p>
    </short-instructions>
  </crowd-bounding-box>
</crowd-form>
```

Note the re-use of `{{ task.input.animal }}` throughout the template. If your manifest had all of the animal names beginning with a capital letter, you could use  `{{ task.input.animal | downcase }}`, incorporating one of Liquid's built-in filters in sentences where it needed to be presented lowercase.

#### **Your manifest file**

Your manifest file should provide the variable values you're using in your template. You can do some transformation of your manifest data in your pre-annotation Lambda, but if you don't need to, you maintain a lower risk of errors and your Lambda will run faster. Here's a sample manifest file for the template.

```
{"source-ref": "<S3 image URI>", "animal": "horse"}
{"source-ref": "<S3 image URI>", "animal" : "bird"}
{"source-ref": "<S3 image URI>", "animal" : "dog"}
{"source-ref": "<S3 image URI>", "animal" : "cat"}
```

## Your pre-annotation Lambda function

As part of the job set-up, provide the ARN of an Amazon Lambda function that can be called to process your manifest entries and pass them to the template engine.

### Naming your Lambda function

The best practice in naming your function is to use one of the following four strings as part of the function name: SageMaker, Sagemaker, sagemaker, or LabelingFunction. This applies to both your pre-annotation and post-annotation functions.

When you're using the console, if you have Amazon Lambda functions that are owned by your account, a drop-down list of functions meeting the naming requirements will be provided to choose one.

In this very basic example, you're just passing through the information from the manifest without doing any additional processing on it. This sample pre-annotation function is written for Python 3.7.

```
import json

def lambda_handler(event, context):
    return {
        "taskInput": event['dataObject']
    }
```

The JSON object from your manifest will be provided as a child of the event object. The properties inside the taskInput object will be available as variables to your template, so simply setting the value of taskInput to event[ 'dataObject' ] will pass all the values from your manifest object to your template without having to copy them individually. If you wish to send more values to the template, you can add them to the taskInput object.

## Your post-annotation Lambda function

As part of the job set-up, provide the ARN of an Amazon Lambda function that can be called to process the form data when a worker completes a task. This can be as simple or complex as you want. If you want to do answer consolidation and scoring as it comes in, you can apply the scoring and/or consolidation algorithms of your choice. If you want to store the raw data for offline processing, that is an option.

### Provide permissions to your post-annotation Lambda

The annotation data will be in a file designated by the s3Uri string in the payload object. To process the annotations as they come in, even for a simple pass through function, you need to assign S3ReadOnly access to your Lambda so it can read the annotation files.

In the Console page for creating your Lambda, scroll to the **Execution role** panel. Select **Create a new role from one or more templates**. Give the role a name. From the **Policy templates** drop-down, choose **Amazon S3 object read-only permissions**. Save the Lambda and the role will be saved and selected.

The following sample is in Python 2.7.

```
import json
import boto3
from urlparse import urlparse

def lambda_handler(event, context):
    consolidated_labels = []

    parsed_url = urlparse(event['payload']['s3Uri']);
    s3 = boto3.client('s3')
    textFile = s3.get_object(Bucket = parsed_url.netloc, Key = parsed_url.path[1:])
    filecont = textFile['Body'].read()
    annotations = json.loads(filecont);

    for dataset in annotations:
```

```

for annotation in dataset['annotations']:
    new_annotation = json.loads(annotation['annotationData']['content'])
    label = {
        'datasetObjectId': dataset['datasetObjectId'],
        'consolidatedAnnotation' : {
            'content': {
                event['labelAttributeName']: {
                    'workerId': annotation['workerId'],
                    'boxesInfo': new_annotation,
                    'imageSource': dataset['dataObject']
                }
            }
        }
    }
    consolidated_labels.append(label)

return consolidated_labels

```

The post-annotation Lambda will often receive batches of task results in the event object. That batch will be the payload object the Lambda should iterate through. What you send back will be an object meeting the [API contract \(p. 307\)](#).

## The output of your labeling job

You'll find the output of the job in a folder named after your labeling job in the target S3 bucket you specified. It will be in a subfolder named `manifests`.

For a bounding box task, the output you find in the output manifest will look a bit like the demo below. The example has been cleaned up for printing. The actual output will be a single line per record.

### Example : JSON in your output manifest

```
{
  "source-ref": "<URL>",
  "<label attribute name>":
  {
    "workerId": "<URL>",
    "imageSource": "<image URL>",
    "boxesInfo": "{\"boundingBox\": {\"boundingBoxes\": [{\"height\": 878, \"label\": \"bird\", \"left\": 208, \"top\": 6, \"width\": 809}], \"inputImageProperties\": {\"height\": 924, \"width\": 1280}}}",
    "<label attribute name>-metadata":
    {
      "type": "groundTruth/custom",
      "job_name": "<Labeling job name>",
      "human-annotated": "yes"
    },
    "animal" : "bird"
  }
}
```

Note how the additional `animal` attribute from your original manifest is passed to the output manifest on the same level as the `source-ref` and labeling data. Any properties from your input manifest, whether they were used in your template or not, will be passed to the output manifest.

## Demo Template: Labeling Intents with `crowd-classifier`

If you choose a custom template, you'll reach the **Custom labeling task panel**. There you can select from multiple starter templates that represent some of the more common tasks. The templates provide a starting point to work from in building your customized labeling task's template.

In this demonstration, you work with the **Intent Detection** template, which uses the [crowd-classifier \(p. 496\)](#) element, and the Amazon Lambda functions needed for processing your data before and after the task.

### Topics

- [Starter Intent Detection custom template \(p. 326\)](#)
- [Your Intent Detection custom template \(p. 326\)](#)
- [Your pre-annotation Lambda function \(p. 330\)](#)
- [Your post-annotation Lambda function \(p. 330\)](#)
- [Your labeling job output \(p. 331\)](#)

## Starter Intent Detection custom template

This is the intent detection template that is provided as a starting point.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-classifier
    name="intent"
    categories="{{ task.input.labels | to_json | escape }}"
    header="Pick the most relevant intention expressed by the below text"
  >
    <classification-target>
      {{ task.input.utterance }}
    </classification-target>

    <full-instructions header="Intent Detection Instructions">
      <p>Select the most relevant intention expressed by the text.</p>
      <div>
        <p><strong>Example: </strong>I would like to return a pair of shoes</p>
        <p><strong>Intent: </strong>Return</p>
      </div>
    </full-instructions>

    <short-instructions>
      Pick the most relevant intention expressed by the text
    </short-instructions>
  </crowd-classifier>
</crowd-form>
```

The custom templates use the [Liquid template language](#), and each of the items between double curly braces is a variable. The pre-annotation Amazon Lambda function should provide an object named `taskInput` and that object's properties can be accessed as `{{ task.input.<property name> }}` in your template.

## Your Intent Detection custom template

In the starter template, there are two variables: the `task.input.labels` property in the `crowd-classifier` element opening tag and the `task.input.utterance` in the `classification-target` region's content.

Unless you need to offer different sets of labels with different utterances, avoiding a variable and just using text will save processing time and creates less possibility of error. The template used in this demonstration will remove that variable, but variables and filters like `to_json` are explained in more detail in the [crowd-bounding-box demonstration](#) article.

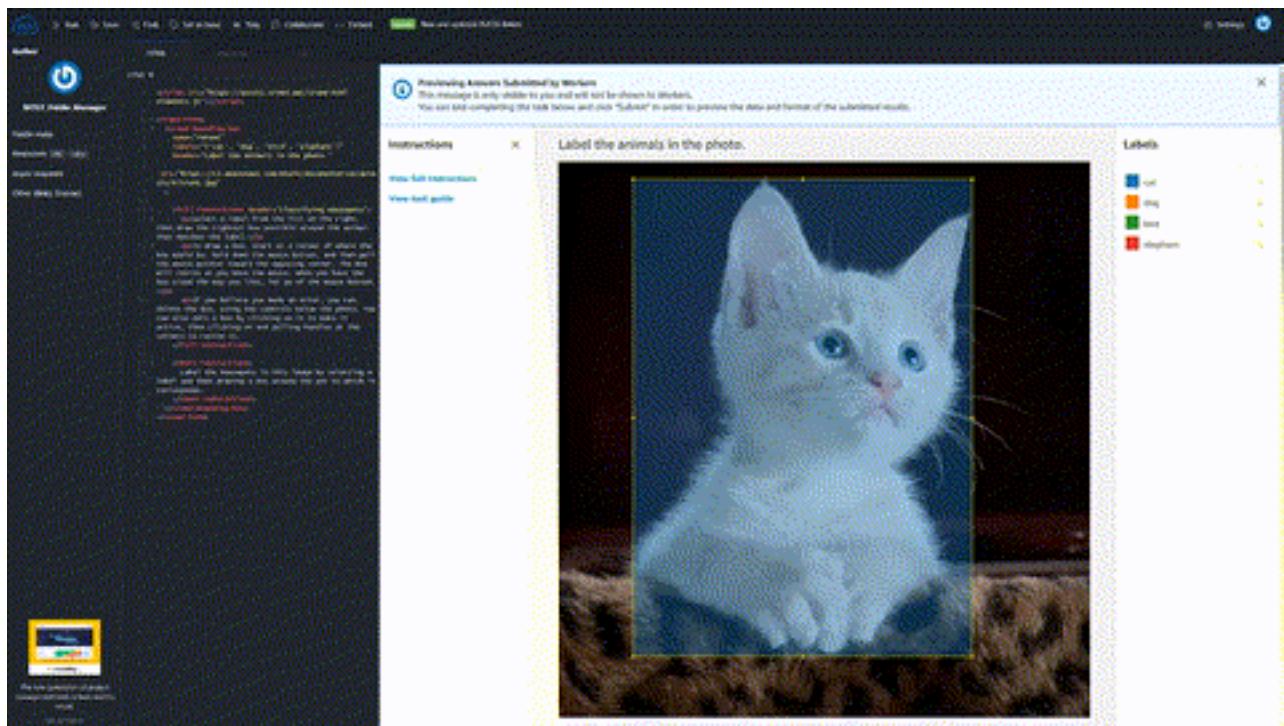
## Styling Your Elements

Two parts of these custom elements that sometimes get overlooked are the `<full-instructions>` and `<short-instructions>` regions. Good instructions generate good results.

In the elements that include these regions, the `<short-instructions>` appear automatically in the "Instructions" pane on the left of the worker's screen. The `<full-instructions>` are linked from the "View full instructions" link near the top of that pane. Clicking the link opens a modal pane with more detailed instructions.

You can not only use HTML, CSS, and JavaScript in these sections, you are encouraged to if you believe you can provide a strong set of instructions and examples that will help workers complete your tasks with better speed and accuracy.

### Example Try out a sample with JSFiddle



Try out an [example `<crowd-classifier>` task](#). The example is rendered by JSFiddle, therefore all the template variables are replaced with hard-coded values. Click the "View full instructions" link to see a set of examples with extended CSS styling. You can fork the project to experiment with your own changes to the CSS, adding sample images, or adding extended JavaScript functionality.

### Example : Final Customized Intent Detection Template

This uses the [example `<crowd-classifier>` task](#), but with a variable for the `<classification-target>`. If you are trying to keep a consistent CSS design among a series of different labeling jobs, you can include an external stylesheet using a `<link rel...>` element the same way you'd do in any other HTML document.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-classifier
    name="intent"
    categories="['buy', 'eat', 'watch', 'browse', 'leave']"
```

```

        header="Pick the most relevant intent expressed by the text below"
    >
    <classification-target>
        {{ task.input.source }}
    </classification-target>

    <full-instructions header="Emotion Classification Instructions">
        <p>In the statements and questions provided in this exercise, what category of action
is the speaker interested in doing?</p>
        <table>
            <tr>
                <th>Example Utterance</th>
                <th>Good Choice</th>
            </tr>
            <tr>
                <td>When is the Seahawks game on?</td>
                <td>
                    eat<br>
                    <greenbg>watch</greenbg>
                    <botchoice>browse</botchoice>
                </td>
            </tr>
            <tr>
                <th>Example Utterance</th>
                <th>Bad Choice</th>
            </tr>
            <tr>
                <td>When is the Seahawks game on?</td>
                <td>
                    buy<br>
                    <greenbg>eat</greenbg>
                    <botchoice>watch</botchoice>
                </td>
            </tr>
        </table>
    </full-instructions>

    <short-instructions>
        What is the speaker expressing they would like to do next?
    </short-instructions>
    </crowd-classifier>
</crowd-form>
<style>
    greenbg {
        background: #feee23;
        display: block;
    }

    table {
        *border-collapse: collapse; /* IE7 and lower */
        border-spacing: 0;
    }

    th, tfoot, .fakehead {
        background-color: #8888ee;
        color: #f3f3f3;
        font-weight: 700;
    }

    th, td, tfoot {
        border: 1px solid blue;
    }

    th:first-child {
        border-radius: 6px 0 0 0;
    }

```

```

th:last-child {
    border-radius: 0 6px 0 0;
}

th:only-child{
    border-radius: 6px 6px 0 0;
}

tfoot:first-child {
    border-radius: 0 0 6px 0;
}

tfoot:last-child {
    border-radius: 0 0 0 6px;
}

tfoot:only-child{
    border-radius: 6px 6px;
}

td {
    padding-left: 15px ;
    padding-right: 15px ;
}

botchoice {
    display: block;
    height: 17px;
    width: 490px;
    overflow: hidden;
    position: relative;
    background: #fff;
    padding-bottom: 20px;
}

botchoice:after {
    position: absolute;
    bottom: 0;
    left: 0;
    height: 100%;
    width: 100%;
    content: "";
    background: linear-gradient(to top,
        rgba(255,255,255, 1) 55%,
        rgba(255,255,255, 0) 100%
    );
    pointer-events: none; /* so the text is still selectable */
}

```

### Example : Your manifest file

If you are preparing your manifest file manually for a text-classification task like this, have your data formatted in the following manner.

```
{"source": "Roses are red"}
{"source": "Violets are Blue"}
 {"source": "Ground Truth is the best"}
 {"source": "And so are you"}
```

This differs from the manifest file used for the "[Demo Template: Annotation of Images with crowd-bounding-box \(p. 321\)](#)" demonstration in that `source-ref` was used as the property name instead

of source. The use of `source-ref` designates S3 URIs for images or other files that must be converted to HTTP. Otherwise, `source` should be used like it is with the text strings above.

## Your pre-annotation Lambda function

As part of the job set-up, provide the ARN of an Amazon Lambda that can be called to process your manifest entries and pass them to the template engine.

This Lambda function is required to have one of the following four strings as part of the function name: `SageMaker`, `Sagemaker`, `sagemaker`, or `LabelingFunction`.

This applies to both your pre-annotation and post-annotation Lambdas.

When you're using the console, if you have Lambdas that are owned by your account, a drop-down list of functions meeting the naming requirements will be provided to choose one.

In this very basic sample, where you have only one variable, it's primarily a pass-through function. Here's a sample pre-labeling Lambda using Python 3.7.

```
import json

def lambda_handler(event, context):
    return {
        "taskInput": event['dataObject']
    }
```

The `dataObject` property of the event contains the properties from a data object in your manifest.

In this demonstration, which is a simple pass through, you just pass that straight through as the `taskInput` value. If you add properties with those values to the `event['dataObject']` object, they will be available to your HTML template as Liquid variables with the format `{task.input.property name}`.

## Your post-annotation Lambda function

As part of the job set up, provide the ARN of an Lambda function that can be called to process the form data when a worker completes a task. This can be as simple or complex as you want. If you want to do answer-consolidation and scoring as data comes in, you can apply the scoring or consolidation algorithms of your choice. If you want to store the raw data for offline processing, that is an option.

### Set permissions for your post-annotation Lambda function

The annotation data will be in a file designated by the `s3Uri` string in the `payload` object. To process the annotations as they come in, even for a simple pass through function, you need to assign `S3ReadOnly` access to your Lambda so it can read the annotation files.

In the Console page for creating your Lambda, scroll to the **Execution role** panel. Select **Create a new role from one or more templates**. Give the role a name. From the **Policy templates** drop-down, choose **Amazon S3 object read-only permissions**. Save the Lambda and the role will be saved and selected.

The following sample is for Python 3.7.

```
import json
import boto3
from urllib.parse import urlparse

def lambda_handler(event, context):
    consolidated_labels = []

    parsed_url = urlparse(event['payload']['s3Uri']);
    s3 = boto3.client('s3')
    textFile = s3.get_object(Bucket = parsed_url.netloc, Key = parsed_url.path[1:])
```

```

filecont = textFile['Body'].read()
annotations = json.loads(filecont);

for dataset in annotations:
    for annotation in dataset['annotations']:
        new_annotation = json.loads(annotation['annotationData']['content'])
        label = {
            'datasetObjectId': dataset['datasetObjectId'],
            'consolidatedAnnotation' : {
                'content': {
                    event['labelAttributeName']: {
                        'workerId': annotation['workerId'],
                        'result': new_annotation,
                        'labeledContent': dataset['dataObject']
                    }
                }
            }
        }
        consolidated_labels.append(label)

return consolidated_labels

```

## Your labeling job output

The post-annotation Lambda will often receive batches of task results in the event object. That batch will be the payload object the Lambda should iterate through.

You'll find the output of the job in a folder named after your labeling job in the target S3 bucket you specified. It will be in a subfolder named `manifests`.

For an intent detection task, the output in the output manifest will look a bit like the demo below. The example has been cleaned up and spaced out to be easier for humans to read. The actual output will be more compressed for machine reading.

### Example : JSON in your output manifest

```

[
  {
    "datasetObjectId": "<Number representing item's place in the manifest>",
    "consolidatedAnnotation":
    {
      "content":
      {
        "<name of labeling job>":
        {
          "workerId": "private.us-east-1.XXXXXXXXXXXXXXXXXXXXXX",
          "result":
          {
            "intent":
            {
              "label": "<label chosen by worker>"
            }
          },
          "labeledContent":
          {
            "content": "<text content that was labeled>"
          }
        }
      }
    }
  },
  "datasetObjectId": "<Number representing item's place in the manifest>",
  "consolidatedAnnotation":
  {

```

```
"content":  
{  
    "<name of labeling job>":  
    {  
        "workerId": "private.us-east-1.6UDLPKQZHYZWJOSCA4MBJBB7FWE",  
        "result":  
        {  
            "intent":  
            {  
                "label": "<label chosen by worker>"  
            },  
            "labeledContent":  
            {  
                "content": "<text content that was labeled>"  
            }  
        }  
    },  
    ...  
    ...  
}  
]
```

This should help you create and use your own custom template.

## Custom Workflows via the API

When you have created your custom UI template (Step 2) and processing Lambda functions (Step 3), you should place the template in an Amazon S3 bucket with a file name format of: <FileName>.liquid.html.

Use the [CreateLabelingJob](#) action to configure your task. You'll use the location of a custom template ([Step 2: Creating your custom worker task template \(p. 301\)](#)) stored in a <filename>.liquid.html file on S3 as the value for the `UiTemplateS3Uri` field in the `UiConfig` object within the `HumanTaskConfig` object.

For the Amazon Lambda tasks described in [Step 3: Processing with Amazon Lambda \(p. 307\)](#), the post-annotation task's ARN will be used as the value for the `AnnotationConsolidationLambdaArn` field, and the pre-annotation task will be used as the value for the `PreHumanTaskLambdaArn`.

## Create a Labeling Job

You can create a labeling job in the Amazon SageMaker console and by using an Amazon SDK in your preferred language to run `CreateLabelingJob`. After a labeling job has been created, you can track worker metrics (for private workforces) and your labeling job status using [CloudWatch](#).

Before you create a labeling job it is recommended that you review the following pages, as applicable:

- You can specify our input data using an automatic data setup in the console, or an input manifest file in either the console or when using `CreateLabelingJob` API. For automated data setup, see [Automated Data Setup \(p. 364\)](#). To learn how to create an input manifest file, see [Use an Input Manifest File \(p. 363\)](#).
- Review labeling job input data quotas: [Input Data Quotas \(p. 371\)](#).

After you have chosen your task type, use the topics on this page to learn how to create a labeling job.

If you are a new Ground Truth user, we recommend that you start by walking through the demo in [Getting started \(p. 166\)](#).

**Important**

Ground Truth requires all S3 buckets that contain labeling job input image data to have a CORS policy attached. To learn more, see [CORS Permission Requirement \(p. 439\)](#).

**Topics**

- [Built-in Task Types \(p. 333\)](#)
- [Creating Instruction Pages \(p. 333\)](#)
- [Create a Labeling Job \(Console\) \(p. 336\)](#)
- [Create a Labeling Job \(API\) \(p. 338\)](#)
- [Create a Streaming Labeling Job \(p. 343\)](#)
- [Create a Labeling Category Configuration File with Label Category and Frame Attributes \(p. 348\)](#)

## Built-in Task Types

Amazon SageMaker Ground Truth has several built-in task types. Ground Truth provides a worker task template for built-in task types. Additionally, some built-in task types support [Automate Data Labeling \(p. 430\)](#). The following topics describe each built-in task type and demo the worker task templates that are provided by Ground Truth in the console. To learn how to create a labeling job in the console using one of these task types, select the task type page.

Label Images	Label Text	Label Videos and Video Frames	Label 3D Point Clouds
<ul style="list-style-type: none"><li>• <a href="#">Bounding Box (p. 171)</a></li><li>• <a href="#">Image Classification (Single Label) (p. 180)</a></li><li>• <a href="#">Image Classification (Multi-label) (p. 183)</a></li><li>• <a href="#">Image Semantic Segmentation (p. 177)</a></li><li>• <a href="#">Verify and Adjust Labels (p. 293)</a></li></ul>	<ul style="list-style-type: none"><li>• <a href="#">Named Entity Recognition (p. 188)</a></li><li>• <a href="#">Text Classification (Single Label) (p. 191)</a></li><li>• <a href="#">Text Classification (Multi-label) (p. 194)</a></li></ul>	<ul style="list-style-type: none"><li>• <a href="#">Video Classification (p. 197)</a></li><li>• <a href="#">Video Frame Object Detection (p. 202)</a></li><li>• <a href="#">Video Frame Object Tracking (p. 206)</a></li></ul>	<ul style="list-style-type: none"><li>• <a href="#">3D Point Cloud Object Detection (p. 234)</a></li><li>• <a href="#">3D Point Cloud Object Tracking (p. 241)</a></li><li>• <a href="#">3D Point Cloud Semantic Segmentation (p. 250)</a></li></ul>

**Note**

Each of the video frame and 3D point cloud task types has an *adjustment* task type that you use to verify and adjust labels from a previous labeling job. Select a video frame or 3D point cloud task type page above to learn how to adjust labels created using that task type.

## Creating Instruction Pages

Create custom instructions for labeling jobs to improve your worker's accuracy in completing their task. You can modify the default instructions that are provided in the console or you can create your own. The instructions are shown to the worker on the page where they complete their labeling task.

There are two kinds of instructions:

- *Short instructions*—instructions that are shown on the same webpage where the worker completes their task. These instructions should provide an easy reference to show the worker the correct way to label an object.

- **Full instructions**—instructions that are shown on a dialog box that overlays the page where the worker completes their task. We recommend that you provide detailed instructions for completing the task with multiple examples showing edge cases and other difficult situations for labeling objects.

Create instructions in the console when you are creating your labeling job. Start with the existing instructions for the task and use the editor to modify them to suit your labeling job.

#### Note

Once you create your labeling job, it will automatically start and you will not be able to modify your worker instructions. If you need to change your worker instructions, stop the labeling job that you created, clone it, and modify your worker instructions before creating a new job.

You can clone a labeling job in the console by selecting the labeling job and then selecting **Clone** in the **Actions** menu.

To clone a labeling job using the Amazon SageMaker API or your preferred Amazon SageMaker SDK, make a new request to the `CreateLabelingJob` operation with the same specifications as your original job after modifying your worker instructions.

## Short Instructions

Short instructions appear on the same web page that workers use to label your data object. For example, the following is the editing page for a bounding box task. The short instructions panel is on the left.

Bounding box labeling tool

Provide labeling instructions with examples below for workers. Workers will be viewing these instructions when they perform your tasks. Make sure the pop-up blocker of the browser is disabled before generating the preview.

Preview

**GOOD EXAMPLE**  
Enter description of a correct bounding box label

Upload image

Add a good example

**BAD EXAMPLE**  
Enter description of an incorrect bounding box label

Upload image

Add a bad example

Enter a brief description of the task



< >

Label  
Add a label name

\* Additional instructions - Optional

Keep in mind that a worker will only spend seconds looking at the short instructions. Workers must be able to scan and understand your information quickly. In all cases it should take less time to understand the instructions than it takes to complete the task. Keep these points in mind:

- Your instructions should be clear and simple.
- Pictures are better than words. Create a simple illustration of your task that your workers can immediately understand.
- If you must use words, use short, concise examples.
- Your short instructions are more important than your full instructions.

The Amazon SageMaker Ground Truth console provides an editor so that you can create your short instructions. Replace the placeholder text and images with instructions for your task. Preview the worker's task page by choosing **Preview**. The preview will open in a new window, be sure to turn off pop-up blocking so that the window will show.

## Full Instructions

You can provide additional instructions for your workers in a dialog box that overlays the page where workers label your data objects. Use full instructions to explain more complex tasks and to show workers the proper way to label edge cases or other difficult objects.

You can create full instructions using an editor in the Ground Truth console. As with quick instructions, keep the following in mind:

- Workers will want detailed instruction the first few times that they complete your task. Any information that they *must* have should be in the quick instructions.
- Pictures are more important than words.
- Text should be concise.
- Full instructions should supplement the short instructions. Don't repeat information that appears in the short instructions.

The Ground Truth console provides an editor so that you can create your full instructions. Replace the placeholder text and images with instructions for your task. Preview the full instruction page by choosing **Preview**. The preview will open in a new window, be sure to turn off pop-up blocking so that the window will show.

## Add example images to your instructions

Images provide useful examples for your workers. To add a publicly accessible image to your instructions:

- Place the cursor where the image should go in the instructions editor.
- Click the image icon in the editor toolbar.
- Enter the URL of your image.

If your instruction image in Amazon S3 is not publicly accessible:

- As the image URL, enter: `{} 'https://s3.amazonaws.com/your-bucket-name/image-file-name' | grant_read_access {}`.
- This renders the image URL with a short-lived, one-time access code appended so the worker's browser can display it. A broken image icon is displayed in the instructions editor, but previewing the tool displays the image in the rendered preview.

## Create a Labeling Job (Console)

You can use the Amazon SageMaker console to create a labeling job for all of the Ground Truth built-in task types and custom labeling workflows. For built-in task types, we recommend that you use this page alongside the [page for your task type](#). Each task type page includes specific details on creating a labeling job using that task type.

You need to provide the following to create a labeling job in the SageMaker console:

- An input manifest file in Amazon S3. You can place your input dataset in Amazon S3 and automatically generate a manifest file using the Ground Truth console (not supported for 3D point cloud labeling jobs).

Alternatively, you can manually create an input manifest file. To learn how, see [Input Data \(p. 363\)](#).

- An Amazon S3 bucket to store your output data.
- An IAM role with permission to access your resources in Amazon S3 and with a SageMaker execution policy attached. For a general solution, you can attach the managed policy, `AmazonSageMakerFullAccess`, to an IAM role and include `sagemaker` in your bucket name.

For more granular policies, see [the section called “IAM Permissions” \(p. 440\)](#).

3D point cloud task types have additional security considerations. [Learn more](#).

- A work team. You create a work team from a workforce made up of Amazon Mechanical Turk workers, vendors, or your own private workers. To learn more, see [Create and Manage Workforces \(p. 456\)](#).

You cannot use the Mechanical Turk workforce for 3D point cloud or video frame labeling jobs.

- If you are using a custom labeling workflow, you must save a worker task template in Amazon S3 and provide an Amazon S3 URI for that template. For more information, see [Step 2: Creating your custom worker task template \(p. 301\)](#).
- (Optional) An Amazon KMS key ARN if you want SageMaker to encrypt the output of your labeling job using your own Amazon KMS encryption key instead of the default Amazon S3 service key.
- (Optional) Existing labels for the dataset you use for your labeling job. Use this option if you want workers to adjust, or approve and reject labels.
- If you want to create an adjustment or verification labeling job, you must have an output manifest file in Amazon S3 that contains the labels you want adjusted or verified. This option is only supported for bounding box and semantic segmentation image labeling jobs and 3D point cloud and video frame labeling jobs. It is recommended that you use the instructions on [Verify and Adjust Labels \(p. 293\)](#) to create a verification or adjustment labeling job.

### Important

Your work team, input manifest file, output bucket, and other resources in Amazon S3 must be in the same Amazon Region you use to create your labeling job.

When you create a labeling job using the SageMaker console, you add worker instructions and labels to the worker UI that Ground Truth provides. You can preview and interact with the worker UI while creating your labeling job in the console. You can also see a preview of the worker UI on your [built-in task type page](#).

### To create a labeling job (console)

1. Sign in to the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the left navigation pane, choose **Labeling jobs**.
3. On the **Labeling jobs** page, choose **Create labeling job**.
4. For **Job name**, enter a name for your labeling job.

5. (Optional) If you want to identify your labels with a key, select **I want to specify a label attribute name different from the labeling job name**. If you do not select this option, the labeling job name you specified in the previous step will be used to identify your labels in your output manifest file.
6. Choose a data setup to setup to set up a connection between your input dataset and Ground Truth.
  - For **Automated data setup**:
    - Follow the instructions in [Automated Data Setup \(p. 364\)](#) for image, text, and video clip labeling jobs.
    - Follow the instructions in [Automated Video Frame Input Data Setup \(p. 401\)](#) for video frame labeling jobs.
  - For **Manual data setup**:
    - For **Input dataset location**, provide the location in Amazon S3 in which your input manifest file is located. For example, if your input manifest file, manifest.json, is located in **example-bucket**, enter **s3://example-bucket/manifest.json**.
    - For **Output dataset location**, provide the location in Amazon S3 where you want Ground Truth to store the output data from your labeling job.
7. For **IAM Role**, choose an existing IAM role or create an IAM role with permission to access your resources in Amazon S3, to write to the output Amazon S3 bucket specified above, and with a SageMaker execution policy attached.
8. (Optional) For **Additional configuration**, you can specify how much of your dataset you want workers to label, and if you want SageMaker to encrypt the output data for your labeling job using an Amazon KMS encryption key. To encrypt your output data, you must have the required Amazon KMS permissions attached to the IAM role you provided in the previous step. For more details, see [the section called "IAM Permissions" \(p. 440\)](#).
9. In the **Task type** section, under **Task category**, use the dropdown list to select your task category.
10. In **Task selection**, choose your task type.
11. (Optional) Provide tags for your labeling job to make it easier to find in the console later.
12. Choose **Next**.
13. In the **Workers** section, choose the type of workforce you would like to use. For more details about your workforce options see [Create and Manage Workforces \(p. 456\)](#).
14. (Optional) After you've selected your workforce, specify the **Task timeout**. This is the maximum amount of time a worker has to work on a task.

For 3D point cloud annotation tasks, the default task timeout is 3 days. The default timeout for text and image classification and label verification labeling jobs is 5 minutes. The default timeout for all other labeling jobs is 60 minutes.

15. (Optional) For bounding box, semantic segmentation, video frame, and 3D point cloud task types, you can select **Display existing labels** if you want to display labels for your input data set for workers to verify or adjust.

For bounding box and semantic segmentation labeling jobs, this will create an adjustment labeling job.

For 3D point cloud and video frame labeling jobs:

- Select **Adjustment** to create an adjustment labeling job. When you select this option, you can add new labels but you cannot remove or edit existing labels from the previous job. Optionally, you can choose label category attributes and frame attributes that you want workers to edit. To make an attribute editable, select the check box **Allow workers to edit this attribute** for that attribute.

Optionally, you can add new label category and frame attributes.

- Select **Verification** to create an adjustment labeling job. When you select this option, you cannot add, modify, or remove existing labels from the previous job. Optionally, you can choose label

category attributes and frame attributes that you want workers to edit. To make an attribute editable, select the check box **Allow workers to edit this attribute** for that attribute.

We recommend that you can add new label category attributes to the labels that you want workers to verify, or add one or more frame attributes to have workers provide information about the entire frame.

For more information, see [Verify and Adjust Labels \(p. 293\)](#).

16. Configure your workers' UI:

- If you are using a [built-in task type](#), specify workers instructions and labels.
  - For image classification and text classification (single and multi-label) you must specify at least two label categories. For all other built-in task types, you must specify at least one label category.
  - (Optional) If you are creating a 3D point cloud or video frame labeling job, you can specify label category attributes (not supported for 3D point cloud semantic segmentation) and frame attributes. Label category attributes can be assigned to one or more labels. Frame attributes will appear on each point cloud or video frame workers label. To learn more, see [Worker User Interface \(UI\) \(p. 260\)](#) for 3D point cloud and [Worker User Interface \(UI\) \(p. 212\)](#) for video frame.
  - (Optional) Add **Additional instructions** to help your worker complete your task.
  - If you are creating a custom labeling workflow you must :
    - Enter a [custom template](#) in the code box. Custom templates can be created using a combination of HTML, the Liquid templating language and our pre-built web components. Optionally, you can choose a base-template from the drop-down menu to get started.
    - Specify pre-annotation and post-annotation lambda functions. To learn how to create these functions, see [Step 3: Processing with Amazon Lambda \(p. 307\)](#).
17. (Optional) You can select **See preview** to preview your worker instructions, labels, and interact with the worker UI. Make sure the pop-up blocker of the browser is disabled before generating the preview.
18. Choose **Create**.

After you've successfully created your labeling job, you are redirected to the **Labeling jobs** page. The status of the labeling job you just created is **In progress**. This status progressively updates as workers complete your tasks. When all tasks are successfully completed, the status changes to **Completed**.

If an issue occurs while creating the labeling job, its status changes to **Failed**. If one or more data objects

To view more details about the job, choose the labeling job name.

## Next Steps

After your labeling job status changes to **Completed**, you can view your output data in the Amazon S3 bucket that you specified while creating that labeling job. For details about the format of your output data, see [Output Data \(p. 404\)](#).

## Create a Labeling Job (API)

To create a labeling job using the Amazon SageMaker API, you use the [CreateLabelingJob](#) operation. For specific instructions on creating a labeling job for a built-in task type, see that [task type page](#). To learn how to create a streaming labeling job, which is a labeling job that runs perpetually, see [Create a Streaming Labeling Job \(p. 343\)](#).

To use the [CreateLabelingJob](#) operation, you need the following:

- A worker task template (`UiTemplateS3Uri`) or human task UI ARN (`HumanTaskUiArn`) in Amazon S3.
  - For 3D point cloud labeling jobs, use the ARN listed in [HumanTaskUiArn](#) for your task type.
  - If you are using a built-in task type other than 3D point cloud tasks, you can add your worker instructions to one of the pre-built templates and save the template (using a `.html` or `.liquid` extension) in your S3 bucket. Find the pre-build templates on your [task type page](#).
  - If you are using a custom labeling workflow, you can create a custom template and save the template in your S3 bucket. To learn how to built a custom worker template, see [Step 2: Creating your custom worker task template \(p. 301\)](#). For custom HTML elements that you can use to customize your template, see [Crowd HTML Elements Reference \(p. 482\)](#). For a repository of demo templates for a variety of labeling tasks, see [Amazon SageMaker Ground Truth Sample Task UIs](#).
- An input manifest file that specifies your input data in Amazon S3. Specify the location of your input manifest file in `ManifestS3Uri`. For information about creating an input manifest, see [Input Data \(p. 363\)](#). If you create a streaming labeling job, this is optional. To learn how to create a streaming labeling job, see [Create a Streaming Labeling Job \(p. 343\)](#).
- An Amazon S3 bucket to store your output data. You specify this bucket, and optionally, a prefix in `S3OutputPath`.
- A label category configuration file. Each label category name must be unique. Specify the location of this file in Amazon S3 using the `LabelCategoryConfigS3Uri` parameter.

For image classification and text classification (single and multi-label) you must specify at least two label categories. For all other task types, the minimum number of label categories required is one.

For 3D point cloud and video frame task type, use the format in [Create a Labeling Category Configuration File with Label Category and Frame Attributes \(p. 348\)](#).

For all other built-in task types and custom tasks, your label category configuration file must be a JSON file in the following format. Identify the labels you want to use by replacing `label_1`, `label_2`, ..., `label_n` with your label categories.

```
{
  "document-version": "2018-11-28"
  "labels": [
    {"label": "label_1"},
    {"label": "label_2"},
    ...
    {"label": "label_n"}
  ]
}
```

- An Amazon Identity and Access Management (IAM) role with the [AmazonSageMakerGroundTruthExecution](#) managed IAM policy attached and with permissions to access your S3 buckets. Specify this role in `RoleArn`. To learn more about this policy, see [Use IAM Managed Policies with Ground Truth \(p. 440\)](#). If you require more granular permissions, see the section called “[IAM Permissions](#)” (p. 440).

If your input or output bucket name does not contain `sagemaker`, you can attach a policy similar to the following to the role that is passed to the `CreateLabelingJob` operation.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        ...
      ]
    }
  ]
}
```

```
        "arn:aws:s3:::my_input_bucket/*"
    ],
},
{
    "Effect": "Allow",
    "Action": [
        "s3:PutObject"
    ],
    "Resource": [
        "arn:aws:s3:::my_output_bucket/*"
    ]
}
]
```

- A pre-annotation and post-annotation (or annotation-consolidation) Amazon Lambda function Amazon Resource Name (ARN) to process your input and output data.
  - Lambda functions are predefined in each Amazon Region for built-in task types. To find the pre-annotation Lambda ARN for your Region, see [PreHumanTaskLambdaArn](#). To find the annotation-consolidation Lambda ARN for your Region, see [AnnotationConsolidationLambdaArn](#).
  - For custom labeling workflows, you must provide a custom pre- and post-annotation Lambda ARN. To learn how to create these Lambda functions, see [Step 3: Processing with Amazon Lambda \(p. 307\)](#).
- A work team ARN that you specify in `WorkteamArn`. You receive a work team ARN when you subscribe to a vendor workforce or create a private workteam. If you are creating a labeling job for a video frame or point cloud task type, you cannot use the Amazon Mechanical Turk workforce. For all other task types, to use the Mechanical Turk workforce, use the following ARN. Replace `region` with the Amazon Region you are using to create the labeling job.

`arn:aws:sagemaker:region:394669845002:workteam/public-crowd/default`

If you use the [Amazon Mechanical Turk workforce](#), use the `ContentClassifiers` parameter in `DataAttributes` of `InputConfig` to declare that your content is free of personally identifiable information and adult content.

Ground Truth *requires* that your input data is free of personally identifiable information (PII) if you use the Mechanical Turk workforce. If you use Mechanical Turk and do not specify that your input data is free of PII using the `FreeOfPersonallyIdentifiableInformation` flag, your labeling job will fail. Use the `FreeOfAdultContent` flag to declare that your input data is free of adult content. SageMaker may restrict the Amazon Mechanical Turk workers that can view your task if it contains adult content.

To learn more about work teams and workforces, see [Create and Manage Workforces \(p. 456\)](#).

- If you use the Mechanical Turk workforce, you must specify the price you'll pay workers for performing a single task in `PublicWorkforceTaskPrice`.
- To configure the task, you must provide a task description and title using `TaskDescription` and `TaskTitle` respectively. Optionally, you can provide time limits that control how long the workers have to work on an individual task (`TaskTimeLimitInSeconds`) and how long tasks remain in the worker portal, available to workers (`TaskAvailabilityLifetimeInSeconds`).
- (Optional) For [some task types](#), you can have multiple workers label a single data object by inputting a number greater than one for the `NumberOfHumanWorkersPerDataObject` parameter. For more information about annotation consolidation, see [Consolidate Annotations \(p. 429\)](#).
- (Optional) To create an automated data labeling job, specify one of the ARNs listed in `LabelingJobAlgorithmSpecificationArn` in `LabelingJobAlgorithmsConfig`. This ARN identifies the algorithm used in the automated data labeling job. The task type associated with this ARN must match the task type of the `PreHumanTaskLambdaArn` and `AnnotationConsolidationLambdaArn` you specify. Automated data labeling is supported for the following task types: image classification, bounding box, semantic segmentation, and text classification. The minimum number of objects

allowed for automated data labeling is 1,250, and we strongly suggest providing a minimum of 5,000 objects. To learn more about automated data labeling jobs, see [Automate Data Labeling \(p. 430\)](#).

- (Optional) You can provide [StoppingConditions](#) that cause the labeling job to stop if one the conditions is met. You can use stopping conditions to control the cost of the labeling job.

## Examples

The following code examples demonstrate how to create a labeling job using `CreateLabelingJob`. For additional examples, we recommend you use one of the **Ground Truth Labeling Jobs** Jupyter notebooks in the SageMaker Examples section of a SageMaker notebook instance. To learn how to use a notebook example from the SageMaker Examples, see [Example Notebooks \(p. 131\)](#). You can also see these example notebooks on GitHub in the [SageMaker Examples repository](#).

### Amazon SDK for Python (Boto3)

The following is an example of an [Amazon Python SDK \(Boto3\) request](#) to create a labeling job for a built-in task type in the US East (N. Virginia) Region using a private workforce. Replace all *red-italicized text* with your labeling job resources and specifications.

```
response = client.create_labeling_job(
    LabelingJobName="example-labeling-job",
    LabelAttributeName="label",
    InputConfig={
        'DataSource': {
            'S3DataSource': {
                'ManifestS3Uri': "s3://bucket/path/manifest-with-input-data.json"
            }
        },
        'DataAttributes': {
            'ContentClassifiers': [
                "FreeOfPersonallyIdentifiableInformation"|"FreeOfAdultContent",
            ]
        }
    },
    OutputConfig={
        'S3OutputPath': "s3://bucket/path/file-to-store-output-data",
        'KmsKeyId': "string"
    },
    RoleArn="arn:aws:iam::*:role/*",
    LabelCategoryConfigS3Uri="s3://bucket/path/label-categories.json",
    StoppingConditions={
        'MaxHumanLabeledObjectCount': 123,
        'MaxPercentageOfInputDatasetLabeled': 123
    },
    HumanTaskConfig={
        'WorkteamArn': "arn:aws:sagemaker:region::workteam/private-crowd/*",
        'UiConfig': {
            'UiTemplateS3Uri': "s3://bucket/path/custom-worker-task-template.html"
        },
        'PreHumanTaskLambdaArn': "arn:aws:lambda:us-
east-1:432418664414:function:PRE-tasktype",
        'TaskKeywords': [
            "Images",
            "Classification",
            "Multi-label"
        ],
        'TaskTitle': "Multi-label image classification task",
        'TaskDescription': "Select all labels that apply to the images shown",
        'NumberOfHumanWorkersPerDataObject': 1,
        'TaskTimeLimitInSeconds': 3600,
        'TaskAvailabilityLifetimeInSeconds': 21600,
        'MaxConcurrentTaskCount': 1000,
```

```

        'AnnotationConsolidationConfig': {
            'AnnotationConsolidationLambdaArn': "arn:aws:lambda:us-
east-1:432418664414:function:ACS-"
        },
        Tags:[
            {
                'Key': "string",
                'Value': "string"
            },
        ]
    )
)

```

### Amazon CLI

The following is an example of an Amazon CLI request to create a labeling job for a built-in task type in the US East (N. Virginia) Region using the [Amazon Mechanical Turk workforce](#). For more information, see [start-human-loop](#) in the [Amazon CLI Command Reference](#). Replace all ***red-italized text*** with your labeling job resources and specifications.

```

$ aws --region us-east-1 sagemaker create-labeling-job \
--labeling-job-name "example-labeling-job" \
--label-attribute-name "label" \
--role-arn "arn:aws:iam::account-id:role/role-name" \
--input-config '{
    "DataAttributes": {
        "ContentClassifiers": [
            "FreeOfPersonallyIdentifiableInformation",
            "FreeOfAdultContent"
        ]
    },
    "DataSource": {
        "S3DataSource": {
            "ManifestS3Uri": "s3://bucket/path/manifest-with-input-data.json"
        }
    }
}' \
--output-config '{
    "KmsKeyId": "",
    "S3OutputPath": "s3://bucket/path/file-to-store-output-data"
}' \
--human-task-config '{
    "AnnotationConsolidationConfig": {
        "AnnotationConsolidationLambdaArn": "arn:aws:lambda:us-
east-1:432418664414:function:ACS-"
    },
    "TaskAvailabilityLifetimeInSeconds": 21600,
    "TaskTimeLimitInSeconds": 3600,
    "NumberOfHumanWorkersPerDataObject": 1,
    "PreHumanTaskLambdaArn": "arn:aws:lambda:us-
east-1:432418664414:function:PRE-tasktype",
    "WorkteamArn": "arn:aws:sagemaker:us-east-1:394669845002:workteam/public-crowd/
default",
    "PublicWorkforceTaskPrice": {
        "AmountInUsd": {
            "Dollars": 0,
            "TenthFractionsOfACent": 6,
            "Cents": 3
        }
    },
    "TaskDescription": "Select all labels that apply to the images shown",
    "MaxConcurrentTaskCount": 1000,
    "TaskTitle": "Multi-label image classification task",
    "TaskKeywords": [
        "Images"
    ],
    "TaskTemplateArn": "arn:aws:sagemaker:us-east-1:394669845002:task-template/public-crowd/
default"
}'

```

```
        "Classification",
        "Multi-label"
    ],
    "UiConfig": {
        "UiTemplateS3Uri": "s3://bucket/path/custom-worker-task-template.html"
    }
}'
```

For more information about this operation, see [CreateLabelingJob](#). For information about how to use other language-specific SDKs, see [See Also](#) in the [CreateLabelingJobs](#) topic.

## Create a Streaming Labeling Job

Streaming labeling jobs enable you to send individual data objects in real time to a perpetually running, streaming labeling job. To create a streaming labeling job, you must create an Amazon SNS *input topic* and specify this topic in [CreateLabelingJob](#) parameters `InputConfig` of `SnsDataSource`. Optionally, you can also create an Amazon SNS *output topic* and specify it in `OutputConfig` if you want to receive label data in real time.

**Important**

If you are a new user of Ground Truth streaming labeling jobs, it is recommended that you review [Ground Truth Streaming Labeling Jobs \(p. 366\)](#) before creating a streaming labeling job.

Use the following sections to create the resources that you need and can use to create a streaming labeling job:

- Learn how to create SNS topics with the permissions required for Ground Truth streaming labeling jobs by following the steps in [Create Amazon SNS Input and Output Topics \(p. 344\)](#). Your SNS topics must be created in the same Amazon Region as your labeling job.
- See [Subscribe an Endpoint to Your Amazon SNS Output Topic \(p. 345\)](#) to learn how to set up an endpoint to receive labeling task output data at a specified endpoint each time a labeling task is completed.
- To learn how to configure your Amazon S3 bucket to send notifications to your Amazon SNS input topic, see [Set up Amazon S3 Bucket Event Notifications \(p. 346\)](#).
- Optionally, add data objects that you want to have labeled as soon as the labeling job starts to your input manifest. For more information, see [Create a Manifest File \(Optional\) \(p. 346\)](#).
- There are other resources required to create a labeling job, such as an IAM role, Amazon S3 bucket, a worker task template and label categories. These are described in the Ground Truth documentation on creating a labeling job. For more information, see [Create a Labeling Job \(p. 332\)](#).

**Important**

When you create a labeling job you must provide an IAM execution role. Attach the Amazon managed policy **AmazonSageMakerGroundTruthExecution** to this role to ensure it has required permissions to execute your labeling job.

When you submit a request to create a streaming labeling job, the state of your labeling job is `Initializing`. Once the labeling job is active, the state changes to `InProgress`. Do not send new data objects to your labeling job or attempt to stop your labeling job while it is in the `Initializing` state. Once the state changes to `InProgress`, you can start sending new data objects using Amazon SNS and the Amazon S3 configuration.

### Topics

- [Create Amazon SNS Input and Output Topics \(p. 344\)](#)
- [Set up Amazon S3 Bucket Event Notifications \(p. 346\)](#)

- [Create a Manifest File \(Optional\) \(p. 346\)](#)
- [Example: Use SageMaker API To Create Streaming Labeling Job \(p. 346\)](#)
- [Stop a Streaming Labeling Job \(p. 348\)](#)

## Create Amazon SNS Input and Output Topics

You need to create an Amazon SNS input to create a streaming labeling job. Optionally, you may provide an Amazon SNS output topic.

When you create an Amazon SNS topic to use in your streaming labeling job, note down the topic Amazon Resource Name (ARN). The ARN will be the input values for the parameter `SnsTopicArn` in `InputConfig` and `OutputConfig` when you create a labeling job.

### Create an Input Topic

Your input topic is used to send new data objects to Ground Truth. To create an input topic, follow the instructions in [Creating an Amazon SNS topic](#) in the Amazon Simple Notification Service Developer Guide.

Note down your input topic ARN and use it as input for the `CreateLabelingJob` parameter `SnsTopicArn` in `InputConfig`.

### Create an Output Topic

If you provide an output topic, it is used to send notifications when a data object is labeled. When you create a topic, you have the option to add an encryption key. Use this option to add a Amazon Key Management Service customer managed key (CMK) to your topic to encrypt the output data of your labeling job before it is published to your output topic.

To create an output topic, follow the instructions in [Creating an Amazon SNS topic](#) in the Amazon Simple Notification Service Developer Guide.

If you add encryption, you must attach additional permission to the topic. See [Add Encryption to Your Output Topic \(Optional\) \(p. 344\)](#) for more information.

#### Important

To add a CMK to your output topic while creating a topic in the console, do not use the **(Default) alias/aws/sns** option. Select a CMK key that you created.

Note down your input topic ARN and use it in your `CreateLabelingJob` request in the parameter `SnsTopicArn` in `OutputConfig`.

### Add Encryption to Your Output Topic (Optional)

To encrypt messages published to your output topic, you need to provide an Amazon KMS customer managed key (CMK) to your topic. Modify the following policy and add it to your CMK to give Ground Truth permission to encrypt output data before publishing it to your output topic.

Replace `<account_id>` with the ID of the account that you are using to create your topic. To learn how to find your Amazon account ID, see [Finding Your Amazon Account ID](#).

```
{  
    "Id": "key-console-policy",  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Enable IAM User Permissions",  
            "Effect": "Allow",  
            "Action": "kms:Encrypt",  
            "Resource": "*"  
        }  
    ]  
}
```

```

        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::<account_id>:root"
        },
        "Action": "kms:*",
        "Resource": "*"
    },
    {
        "Sid": "Allow access for Key Administrators",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::<account_id>:role/Admin"
        },
        "Action": [
            "kms>Create*",
            "kms:Describe*",
            "kms:Enable*",
            "kms>List*",
            "kms:Put*",
            "kms:Update*",
            "kms:Revoke*",
            "kms:Disable*",
            "kms:Get*",
            "kms>Delete*",
            "kms:TagResource",
            "kms:UntagResource",
            "kms:ScheduleKeyDeletion",
            "kms:CancelKeyDeletion"
        ],
        "Resource": "*"
    }
]
}

```

Additionally, you must modify and add the following policy to the execution role that you use to create your labeling job (the input value for `RoleArn`).

Replace `<account_id>` with the ID of the account that you are using to create your topic. Replace `<region>` with the Amazon Region you are using to create your labeling job. Replace `<key_id>` with your CMK ID.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "sid1",
            "Effect": "Allow",
            "Action": [
                "kms:Decrypt",
                "kms:GenerateDataKey"
            ],
            "Resource": "arn:aws:kms:<region>:<account_id>:key/<key_id>"
        }
    ]
}

```

For more information on creating and securing keys, see [Creating Keys](#) and [Using Key Policies](#) in the Amazon Key Management Service Developer Guide.

### [Subscribe an Endpoint to Your Amazon SNS Output Topic](#)

When a worker completes a labeling job task from a Ground Truth streaming labeling job, Ground Truth uses your output topic to publish output data to one or more endpoints that you specify. To receive

notifications when a worker finishes a labeling task, you must subscribe an endpoint to your Amazon SNS output topic.

To learn how to add endpoints to your output topic, see [Subscribing to an Amazon SNS topic](#) in the *Amazon Simple Notification Service Developer Guide*.

To learn more about the output data format that is published to these endpoints, see [Output Data \(p. 404\)](#).

**Important**

If you do not subscribe an endpoint to your Amazon SNS output topic, you will not receive notifications when new data objects are labeled.

## Set up Amazon S3 Bucket Event Notifications

You can add an event notification to your Amazon S3 bucket using the Amazon S3 console, API, and language specific Amazon SDKs, or the Amazon Command Line Interface. Set up this event to send notifications to the same Amazon SNS input topic that you specify using `SnsTopicArn` in `InputConfig` when you create a labeling job. Do not set up event notifications using the same Amazon S3 location that you specified for `S3OutputPath` in `OutputConfig` – doing so may result in unwanted data objects being processed by Ground Truth for labeling.

You decide the types of events that you want to send to your Amazon SNS topic. Ground Truth creates a labeling job when you send [object creation events](#).

The event structure sent to your Amazon SNS input topic must be a JSON message formatted using the same structure found in [Event message structure](#).

To see examples of how you can set up an event notification for your Amazon S3 bucket using the Amazon S3 console, Amazon SDK for .NET, and Amazon SDK for Java, follow this walkthrough, [Walkthrough: Configure a bucket for notifications \(SNS topic or SQS queue\)](#) in the Amazon Simple Storage Service Developer Guide.

## Create a Manifest File (Optional)

When you create a streaming labeling job, you have the one time option to add objects (such as images or text) to an input manifest file that you specify in `ManifestS3Uri` of `CreateLabelingJob`. When the streaming labeling job starts, these objects are sent to workers or added to the Amazon SQS queue if the total number of objects exceed `MaxConcurrentTaskCount`. The results are added to the Amazon S3 path that you specify when creating the labeling job periodically as workers complete labeling tasks. Output data is sent to any endpoint that you subscribe to your output topic.

If you want to provide initial objects to be labeled, create a manifest file that identifies these objects and place it in Amazon S3. Specify the S3 URI of this manifest file in `ManifestS3Uri` within `InputConfig`.

To learn how to format your manifest file, see [Input Data \(p. 363\)](#). To use the SageMaker console to automatically generate a manifest file (not supported for 3D point cloud task types), see [Automated Data Setup \(p. 364\)](#).

## Example: Use SageMaker API To Create Streaming Labeling Job

The following is an example of an [Amazon Python SDK \(Boto3\) request](#) that you can use to start a streaming labeling job for a built-in task type in the US East (N. Virginia) Region. For more details about each parameter below see [CreateLabelingJob](#). To learn how you can create a labeling job using this API and associated language specific SDKs, see [Create a Labeling Job \(API\)](#).

In this example, note the following parameters:

- **SnsDataSource** – This parameter appears in `InputConfig` and `OutputConfig` and is used to identify your input and output Amazon SNS topics respectively. To create a streaming labeling job, you are required to provide an Amazon SNS input topic. Optionally, you can also provide an Amazon SNS output topic.
- **S3DataSource** – This parameter is optional. Use this parameter if you want to include an input manifest file of data objects that you want labeled as soon as the labeling job starts.
- **StoppingConditions** – This parameter is ignored when you create a streaming labeling job. To learn more about stopping a streaming labeling job, see [Stop a Streaming Labeling Job \(p. 348\)](#).
- Streaming labeling jobs do not support automated data labeling. Do not include the `LabelingJobAlgorithmsConfig` parameter.

```

response = client.create_labeling_job(
    LabelingJobName= 'example-labeling-job',
    LabelAttributeName='label',
    InputConfig={
        'DataSource': {
            'S3DataSource': {
                'ManifestS3Uri': 's3://bucket/path/manifest-with-input-data.json'
            },
            'SnsDataSource': {
                'SnsTopicArn': 'arn:aws:sns:us-east-1:123456789012:your-sns-input-topic'
            }
        },
        'DataAttributes': {
            'ContentClassifiers': [
                'FreeOfPersonallyIdentifiableInformation' | 'FreeOfAdultContent',
            ]
        }
    },
    OutputConfig={
        'S3OutputPath': 's3://bucket/path/file-to-store-output-data',
        'KmsKeyId': 'string',
        'SnsTopicArn': 'arn:aws:sns:us-east-1:123456789012:your-sns-output-topic'
    },
    RoleArn='arn:aws:iam::*:role/*',
    LabelCategoryConfigS3Uri='s3://bucket/path/label-categories.json',
    HumanTaskConfig={
        'WorkteamArn': 'arn:aws:sagemaker:us-east-1::*:workteam/private-crowd/*',
        'UiConfig': {
            'UiTemplateS3Uri': 's3://bucket/path/custom-worker-task-template.html'
        },
        'PreHumanTaskLambdaArn': 'arn:aws:lambda:us-
east-1:432418664414:function:PRE-tasktype,
        'TaskKeywords': [
            'Example key word',
        ],
        'TaskTitle': 'Multi-label image classification task',
        'TaskDescription': 'Select all labels that apply to the images shown',
        'NumberOfHumanWorkersPerDataObject': 123,
        'TaskTimeLimitInSeconds': 123,
        'TaskAvailabilityLifetimeInSeconds': 123,
        'MaxConcurrentTaskCount': 123,
        'AnnotationConsolidationConfig': {
            'AnnotationConsolidationLambdaArn': 'arn:aws:lambda:us-
east-1:432418664414:function:ACS-tasktype'
        },
        Tags=[
        {
            'Key': 'string',
            'Value': 'string'
        },
    ]
}

```

## Stop a Streaming Labeling Job

You can manually stop your streaming labeling job using the operation [StopLabelingJob](#).

If your labeling job remains idle for over 10 days, it is automatically stopped by Ground Truth. In this context, a labeling job is considered *idle* if no objects are sent to the Amazon SNS input topic and no objects remain in your Amazon SQS queue, waiting to be labeled. For example, if no data objects are fed to the Amazon SNS input topic and all the objects fed to the labeling job are already labeled, Ground Truth starts a timer. After the timer starts, if no items are received within a 10 day period, the labeling job is stopped.

If this 10 day limit poses a problem for your business use case, please contact [Amazon Support](#).

When a labeling job is stopped, its status is STOPPING while Ground Truth cleans up labeling job resources and unsubscribes your Amazon SNS topic from your Amazon SQS queue. The Amazon SQS is *not* deleted by Ground Truth because this queue may contain unprocessed data objects. You should manually delete the queue if you want to avoid incurring additional charges from Amazon SQS. To learn more, see [Amazon SQS pricing](#).

## Create a Labeling Category Configuration File with Label Category and Frame Attributes

When you create a 3D point cloud or video frame labeling job using the Amazon SageMaker API operation `CreateLabelingJob`, you use a label category configuration file to specify your labels and worker instructions. Optionally, you can also provide the following in your label category attribute file:

- You can provide *label category attributes* for video frame and 3D point cloud object tracking and object detection task types. Workers can use one or more attributes to give more information about an object. For example, you may want to use the attribute *occluded* to have workers identify when an object is partially obstructed. You can either specify a label category attribute for a single label using the `categoryAttributes` parameter, or for all labels using the `categoryGlobalAttributes` parameter.
- You can provide *frame attributes* for video frame and 3D point cloud object tracking and object detection task types using `frameAttributes`. When you create a frame attribute, it appears on each frame or point cloud in the worker task. In video frame labeling jobs, these are attributes that workers assign to an entire video frame. For 3D point cloud labeling jobs, these attributes are applied to a single point cloud. Use frame attributes to have workers provide more information about the scene in a specific frame or point cloud.
- For video frame labeling jobs, you use the label category configuration file to specify the task type (bounding box, polyline, polygon, or keypoint) sent to workers.

For workers, specifying values for label category attributes and frame attributes will be optional.

### Important

You should only provide a label attribute name in `auditLabelAttributeName` if you are running an audit job to verify or adjust labels. Use this parameter to input the [LabelAttributeName](#) used in the labeling job that generated the annotations you want your worker to adjust. When you create a labeling job in the console, if you did not specify a label attribute name, the **Name** of your job is used as the `LabelAttributeName`.

### Topics

- [Label Category Configuration File Schema \(p. 349\)](#)
- [Example: Label Category Configuration Files for 3D Point Cloud Labeling Jobs \(p. 354\)](#)
- [Example: Label Category Configuration Files for Video Frame Labeling Jobs \(p. 358\)](#)

- [Creating Worker Instructions \(p. 362\)](#)

## Label Category Configuration File Schema

The following table lists elements you can and must include in your label category configuration file.

**Note**

The parameter `annotationType` is only supported for video frame labeling jobs.

Parameter	Required	Accepted Values	Description
<code>frameAttributes</code>	No	<p>A list of JSON objects.</p> <p><b>Required Parameters in each JSON Object:</b></p> <p><code>name</code>, <code>type</code>, <code>description</code></p> <p><code>minimum</code> and <code>maximum</code> are required if <code>type</code> is "number"</p> <p><b>Optional Parameters in each JSON Object:</b></p> <p><code>enum</code>, <code>editsAllowed</code></p>	<p>Use this parameter to create a frame attribute that is applied to all frames or 3D point clouds in your labeling job.</p> <p>See the third table in this section for more information.</p>
<code>categoryGlobalAttributes</code>	No	<p>A list of JSON objects.</p> <p><b>Required Parameters in each JSON Object:</b></p> <p><code>name</code>, <code>type</code></p> <p><code>minimum</code> and <code>maximum</code> are required if <code>type</code> is "number"</p> <p><b>Optional Parameters in each JSON Object:</b></p> <p><code>description</code>, <code>enum</code>, <code>editsAllowed</code></p>	<p>Use this parameter to create label category attributes that are applied to all labels you specify in <code>labels</code>.</p> <p>See the third table in this section for more information.</p>
<code>labels</code>	Yes	<p>A list of up to 30 JSON objects</p> <p><b>Required Parameters in each JSON Object:</b></p> <p><code>label</code></p> <p><b>Optional Parameters in each JSON Object:</b></p> <p><code>categoryAttributes</code>, <code>editsAllowed</code></p>	<p>Use this parameter to specify your labels, or classes. Add one label for each class.</p> <p>To add a label category attribute to a label, add <code>categoryAttributes</code> to that label.</p> <p>Use <code>editsAllowed</code> to specify whether or not a label can be edited in an adjustment labeling job. Set <code>editsAllowed</code> to</p>

Parameter	Required	Accepted Values	Description
			"none" for verification labeling jobs.  See the following table for more information.
annotationType (only supported for video frame labeling jobs)	No	<p>String</p> <p><b>Accepted Parameters:</b> BoundingBox, Polyline, Polygon, Keypoint</p> <p><b>Default:</b> BoundingBox</p>	<p>Use this to specify the task type for your video frame labeling jobs. For example, for a polygon video frame object detection task, choose Polygon.</p> <p>If you do not specify an annotationType when you create a video frame labeling job, Ground Truth will use BoundingBox by default.</p>
instructions	No	<p>A JSON object</p> <p><b>Required Parameters in each JSON Object:</b></p> <ul style="list-style-type: none"> <li>"shortInstruction",</li> <li>"fullInstruction"</li> </ul>	<p>Use this parameter to add worker instructions to help your workers complete their tasks. For more information about worker instructions, see <a href="#">Worker Instructions (p. 261)</a>.</p> <p>Short instructions must be under 255 characters and long instruction must be under 2,048 characters.</p> <p>For more information, see <a href="#">Creating Worker Instructions (p. 362)</a>.</p>
auditLabelAttributeName	Required for adjustment and verification task types	<p>String</p>	<p>Enter the <a href="#">LabelAttributeName</a> used in the labeling job you want to adjust annotations of.</p> <p>Only use this parameter if you are creating an adjustment job for video frame and 3D point cloud object detection, object tracking, or 3D point cloud semantic segmentation.</p>

The following table describes the parameters that you can and must use to create a list of `Labels`. Each parameter should be included in a JSON object.

Parameter	Required	Accepted Values	Description
<code>label</code>	Yes	String	The name of the label category that is displayed to workers. Each label category name must be unique.
<code>categoryAttributes</code>	No	<p>A list of JSON objects.</p> <p><b>Required Parameters in each JSON Object:</b></p> <ul style="list-style-type: none"> <li><code>name</code>, <code>type</code></li> <li><code>minimum</code> and <code>maximum</code> required if <code>type</code> is "number"</li> </ul> <p><b>Optional Parameters in each JSON Object:</b></p> <ul style="list-style-type: none"> <li><code>description</code>, <code>enum</code>, <code>editsAllowed</code></li> </ul>	<p>Use this parameter to add label category attributes to specific labels you specify in <code>labels</code>.</p> <p>To add one or more label category attributes to a label, include the <code>categoryAttributes</code> JSON object in the same <code>labels</code> JSON object as that label. See the following table for more information.</p>
<code>editsAllowed</code>	No	<p>String</p> <p><b>Supported Values:</b></p> <ul style="list-style-type: none"> <li>"none": no modifications are not allowed.</li> <li>or</li> <li>"any" (Default): all modifications are allowed.</li> </ul>	<p>Specifies whether or not a label can be edited by workers.</p> <p>For video frame or 3D point cloud <i>adjustment</i> labeling jobs, add this parameter to one or more JSON objects in the <code>labels</code> list to specify whether or not a worker can edit a label.</p> <p>For 3D point cloud and video frame <i>verification</i> labeling jobs, add this parameter with the value "none" to each JSON object in the <code>labels</code> list. This will make all labels uneditable.</p>

The following table describes the parameters that you can and must use to create a frame attributes using `frameAttributes` and label category attribute using the `categoryGlobalAttributes` and `categoryAttributes` parameters.

Parameter	Required	Accepted Values	Description
name	Yes	String	<p>Use this parameter to assign a name to your label category or frame attribute. This is the attribute name that workers see.</p> <p>Each label category attribute name in your label category configuration file must be unique. Global label category attributes and label specific label category attributes cannot have the same name.</p>
type	Yes	<p>String</p> <p><b>Required Values:</b></p> <p>"string" or "number"</p>	<p>Use this parameter to define the label category or frame attribute type.</p> <p>If you specify "string" for type and provide an enum value for this attribute, workers will be able to choose from one of the choices you provide.</p> <p>If you specify "string" for type and do not provide an enum value, workers can enter free form text.</p> <p>If you specify number for type, worker can enter a number between the minimum and maximum numbers you specify.</p>
enum	Yes	List of strings	Use this parameter to define options that workers can choose from for this label category or frame attribute. Workers can choose one value specified in enum. For example, if you specify [ "foo", "buzz", "bar"] for enum,

Parameter	Required	Accepted Values	Description
			workers can choose one of foo, buzz, or bar.  You must specify "string" for type to use an enum list.
description	frameAttributes: Yes  categoryAttributes or categoryGlobalAttributes: No	String	Use this parameter to add a description of the label category or frame attribute. You can use this field to give workers more information about the attribute.  This field is only required for frame attributes.
minimum and maximum	Required if attribute type is "number"	Integers	Use these parameters to specify minimum and maximum (inclusive) values workers can enter for numeric label category or frame attributes.  You must specify "number" for type to use minimum and maximum.
editsAllowed	No	String  <b>Required Values:</b> "none": no modifications are not allowed.  or  "any" (Default): all modifications are allowed.	Specifies whether or not a label category or frame attribute can be edited by workers.  For video frame or 3D point cloud <i>adjustment</i> and <i>verification</i> labeling jobs, add this parameter to label category and frame attribute JSON objects to specify whether or not a worker can edit an attribute.

### Label and label category attribute quotas

You can specify up to 10 label category attributes per class. This 10-attribute quotas includes global label category attributes. For example, if you create four global label category attributes, and then assign three label category attributes to label x, that label will have  $4+3=7$  label category attributes in total. For all label category and label category attribute limits, refer to the following table.

Type	Min	Max
Labels (Labels)	1	30
Label name character quota	1	16
Label category attributes per label (sum of <code>categoryAttributes</code> and <code>categoryGlobalAttributes</code> )	0	10
Free form text entry label category attributes per label (sum of <code>categoryAttributes</code> and <code>categoryGlobalAttributes</code> ).	0	5
Frame attributes	0	10
Free form text entry attributes in <code>frameAttributes</code> .	0	5
Attribute name character quota ( <code>name</code> )	1	16
Attribute description character quota ( <code>description</code> )	0	128
Attribute type characters quota ( <code>type</code> )	1	16
Allowed values in the <code>enum</code> list for a <code>string</code> attribute	1	10
Character quota for a value in <code>enum</code> list	1	16
Maximum characters in free form text response for free form text <code>frameAttributes</code>	0	1000
Maximum characters in free form text response for free form text <code>categoryAttributes</code> and <code>categoryGlobalAttributes</code>	0	80

## Example: Label Category Configuration Files for 3D Point Cloud Labeling Jobs

Select a tab in the following tables to see examples of 3D point cloud label category configuration files for object detection, object tracking, semantic segmentation, adjustment, and verification labeling jobs.

### 3D Point Cloud Object Tracking and Object Detection

The following is an example of a label category configuration file that includes label category attributes for a 3D point cloud object detection or object tracking labeling job. This example includes a two frame attributes, which will be added to all point clouds submitted to the labeling job. The `Car` label will include four label category attributes—`x`, `y`, `z`, and the global attribute, `w`.

```
{
```

```

"documentVersion": "2020-03-01",
"frameAttributes": [
    {
        "name": "count players",
        "description": "How many players to you see in the scene?",
        "type": "number"
    },
    {
        "name": "select one",
        "description": "describe the scene",
        "type": "string",
        "enum": ["clear", "blurry"]
    },
],
"categoryGlobalAttributes": [
    {
        "name": "W",
        "description": "label-attributes-for-all-labels",
        "type": "string",
        "enum": ["foo", "buzz", "biz"]
    }
],
"labels": [
    {
        "label": "Car",
        "categoryAttributes": [
            {
                "name": "X",
                "description": "enter a number",
                "type": "number",
            },
            {
                "name": "Y",
                "description": "select an option",
                "type": "string",
                "enum": ["y1", "y2"]
            },
            {
                "name": "Z",
                "description": "submit a free-form response",
                "type": "string",
            }
        ]
    },
    {
        "label": "Pedestrian",
        "categoryAttributes": [...]
    }
],
"instructions": {"shortInstruction": "Draw a tight Cuboid", "fullInstruction": "<html><body><input type='checkbox' value='1'> Draw a tight Cuboid</body></html>"}
}

```

### 3D Point Cloud Semantic Segmentation

The following is an example of a label category configuration file for a 3D point cloud semantic segmentation labeling job.

Label category attributes are not supported for 3D point cloud semantic segmentation task types. Frame attributes are supported. If you provide label category attributes for a semantic segmentation labeling job, they will be ignored.

```
{
    "documentVersion": "2020-03-01",
```

```

"frameAttributes": [
    {
        "name": "count players",
        "description": "How many players to you see in the scene?",
        "type": "number"
    },
    {
        "name": "select one",
        "description": "describe the scene",
        "type": "string",
        "enum": ["clear", "blurry"]
    },
],
"labels": [
    {
        "label": "Car",
    },
    {
        "label": "Pedestrian",
    },
    {
        "label": "Cyclist",
    }
],
"instructions": {"shortInstruction": "Select the appropriate label and
paint all objects in the point cloud that it applies to the same color",
"fullInstruction": "<html markup>"}
}

```

Select a tab in the following table to see an example of a label category configuration file for 3D point cloud verification or adjustment labeling jobs.

#### 3D Point Cloud Adjustment

The following is an example of a label category configuration file for a 3D point cloud object detection or object tracking adjustment labeling job. For 3D point cloud semantic segmentation adjustment labeling jobs, `categoryGlobalAttributes` and `categoryAttributes` are not supported.

You must include `auditLabelAttributeName` to specify the label attribute name of the previous labeling job that you use to create the adjustment labeling job. Optionally, you can use the `editsAllowed` parameter to specify whether or not a label or frame attribute can be edited.

```

{
    "documentVersion": "2020-03-01",
    "frameAttributes": [
        {
            "name": "count players",
            "description": "How many players to you see in the scene?",
            "type": "number"
        },
        {
            "name": "select one",
            "editsAllowed": "none",
            "description": "describe the scene",
            "type": "string",
            "enum": ["clear", "blurry"]
        },
    ],
    "categoryGlobalAttributes": [
        {
            "name": "W",
            "editsAllowed": "any",

```

```

        "description": "label-attributes-for-all-labels",
        "type": "string",
        "enum": [ "foo", "buzz", "biz"]
    }
],
"labels": [
{
    "label": "Car",
    "editsAllowed": "any",
    "categoryAttributes": [
        {
            "name": "X",
            "description": "enter a number",
            "type": "number"
        },
        {
            "name": "Y",
            "description": "select an option",
            "type": "string",
            "enum": ["y1", "y2"],
            "editsAllowed": "any"
        },
        {
            "name": "Z",
            "description": "submit a free-form response",
            "type": "string",
            "editsAllowed": "none"
        }
    ]
},
{
    "label": "Pedestrian",
    "categoryAttributes": [...]
}
],
"instructions": { "shortInstruction": "Draw a tight Cuboid", "fullInstruction": "<html><body><h1>Draw a tight Cuboid</h1></body></html>" },
// include auditLabelAttributeName for label adjustment jobs
"auditLabelAttributeName": "myPrevJobLabelAttributeName"
}

```

### 3D Point Cloud Verification

The following is an example of a label category configuration file you may use for a 3D point cloud object detection or object tracking verification labeling job. For a 3D point cloud semantic segmentation verification labeling job, `categoryGlobalAttributes` and `categoryAttributes` are not supported.

You must include `auditLabelAttributeName` to specify the label attribute name of the previous labeling job that you use to create the verification labeling job. Additionally, you must use the `editsAllowed` parameter to specify that no labels can be edited.

```

{
    "documentVersion": "2020-03-01",
    "frameAttributes": [
        {
            "name": "count players",
            "editsAllowed": "any",
            "description": "How many players do you see in the scene?",
            "type": "number"
        },
        {
            "name": "select one",
            "editsAllowed": "any",

```

```

        "description": "describe the scene",
        "type": "string",
        "enum": [ "clear", "blurry" ]
    },
],
"categoryGlobalAttributes": [
{
    "name": "W",
    "editsAllowed": "none",
    "description": "label-attributes-for-all-labels",
    "type": "string",
    "enum": [ "foo", "buzz", "biz" ]
}
],
"labels": [
{
    "label": "Car",
    "editsAllowed": "none",
    "categoryAttributes": [
{
        "name": "X",
        "description": "enter a number",
        "type": "number",
        "editsAllowed": "none"
},
{
        "name": "Y",
        "description": "select an option",
        "type": "string",
        "enum": [ "y1", "y2" ],
        "editsAllowed": "any"
},
{
        "name": "Z",
        "description": "submit a free-form response",
        "type": "string",
        "editsAllowed": "none"
}
],
{
    "label": "Pedestrian",
    "editsAllowed": "none",
    "categoryAttributes": [ ... ]
}
],
"instructions": { "shortInstruction": "Draw a tight Cuboid", "fullInstruction": "<html><mark>" },
// include auditLabelAttributeName for label verification jobs
"auditLabelAttributeName": "myPrevJobLabelAttributeName"
}
]

```

## Example: Label Category Configuration Files for Video Frame Labeling Jobs

The annotation tools available to your worker and task type used depends on the value you specify for `annotationType`. For example, if you want workers to use key points to track changes in the pose of specific objects across multiple frames, you would specify `Keypoint` for the `annotationType`. If you do not specify an annotation type, `BoundingBox` will be used by default.

The following is an example of a video frame keypoint label category configuration file with label category attributes. This example includes two frame attributes, which will be added to all frames submitted to the labeling job. The `Car` label will include four label category attributes—`x`, `y`, `z`, and the global attribute, `w`.

```
{
    "documentVersion": "2020-03-01",
    "frameAttributes": [
        {
            "name": "count players",
            "description": "How many players do you see in the scene?",
            "type": "number"
        },
        {
            "name": "select one",
            "description": "describe the scene",
            "type": "string",
            "enum": ["clear", "blurry"]
        },
    ],
    "categoryGlobalAttributes": [
        {
            "name": "W",
            "description": "label-attributes-for-all-labels",
            "type": "string",
            "enum": ["foo", "buz", "buzz2"]
        }
    ],
    "labels": [
        {
            "label": "Car",
            "categoryAttributes": [
                {
                    "name": "X",
                    "description": "enter a number",
                    "type": "number"
                },
                {
                    "name": "Y",
                    "description": "select an option",
                    "type": "string",
                    "enum": ["y1", "y2"]
                },
                {
                    "name": "Z",
                    "description": "submit a free-form response",
                    "type": "string"
                }
            ]
        },
        {
            "label": "Pedestrian",
            "categoryAttributes": [...]
        }
    ],
    "annotationType": "Keypoint",
    "instructions": {"shortInstruction": "add example short instructions here",
    "fullInstruction": "<html markup>"}
}
```

Select a tab from the following table to see examples of label category configuration files for video frame adjustment and verification labeling jobs.

#### Video Frame Adjustment

The following is an example of a label category configuration file you may use for a video frame adjustment labeling job.

You must include `auditLabelAttributeName` to specify the label attribute name of the previous labeling job that you use to create the verification labeling job. Optionally, you can use the `editsAllowed` parameter to specify whether or not labels, label category attributes, or frame attributes can be edited.

```
{
    "documentVersion": "2020-03-01",
    "frameAttributes": [
        {
            "name": "count players",
            "editsAllowed": "none",
            "description": "How many players do you see in the scene?",
            "type": "number"
        },
        {
            "name": "select one",
            "description": "describe the scene",
            "type": "string",
            "enum": ["clear", "blurry"]
        },
    ],
    "categoryGlobalAttributes": [
        {
            "name": "W",
            "editsAllowed": "any",
            "description": "label-attributes-for-all-labels",
            "type": "string",
            "enum": ["foo", "buz", "buz2"]
        }
    ],
    "labels": [
        {
            "label": "Car",
            "editsAllowed": "any",
            "categoryAttributes": [
                {
                    "name": "X",
                    "description": "enter a number",
                    "type": "number",
                    "editsAllowed": "any"
                },
                {
                    "name": "Y",
                    "description": "select an option",
                    "type": "string",
                    "enum": ["y1", "y2"],
                    "editsAllowed": "any"
                },
                {
                    "name": "Z",
                    "description": "submit a free-form response",
                    "type": "string",
                    "editsAllowed": "none"
                }
            ]
        },
        {
            "label": "Pedestrian",
            "editsAllowed": "none",
            "categoryAttributes": [...]
        }
    ],
    "annotationType": "Keypoint",
    "instructions": {"shortInstruction": "add example short instructions here",
    "fullInstruction": "<html markup>"}
}
```

```
// include auditLabelAttributeName for label adjustment jobs
"auditLabelAttributeName": "myPrevJobLabelAttributeName"
}
```

## Video Frame Verification

The following is an example of a label category configuration file for a video frame labeling job.

You must include auditLabelAttributeName to specify the label attribute name of the previous labeling job that you use to create the verification labeling job. Additionally, you must use the editsAllowed parameter to specify that no labels can be edited.

```
{
    "documentVersion": "2020-03-01",
    "frameAttributes": [
        {
            "name": "count_players",
            "editsAllowed": "none",
            "description": "How many players do you see in the scene?",
            "type": "number"
        },
        {
            "name": "select_one",
            "editsAllowed": "any",
            "description": "Describe the scene",
            "type": "string",
            "enum": ["clear", "blurry"]
        }
    ],
    "categoryGlobalAttributes": [
        {
            "name": "W",
            "editsAllowed": "none",
            "description": "label-attributes-for-all-labels",
            "type": "string",
            "enum": ["foo", "buz", "buz2"]
        }
    ],
    "labels": [
        {
            "label": "Car",
            "editsAllowed": "none",
            "categoryAttributes": [
                {
                    "name": "X",
                    "description": "enter a number",
                    "type": "number",
                    "editsAllowed": "any"
                },
                {
                    "name": "Y",
                    "description": "select an option",
                    "type": "string",
                    "enum": ["y1", "y2"],
                    "editsAllowed": "any"
                },
                {
                    "name": "Z",
                    "description": "submit a free-form response",
                    "type": "string",
                    "editsAllowed": "none"
                }
            ]
        },
    ],
}
```

```
{  
    "label": "Pedestrian",  
    "editsAllowed": "none",  
    "categoryAttributes": [...]  
}  
],  
"annotationType": "Keypoint",  
"instructions": {"shortInstruction": "add example short instructions here",  
"fullInstruction": "<html markup>"},  
// include auditLabelAttributeName for label adjustment jobs  
"auditLabelAttributeName": "myPrevJobLabelAttributeName"  
}
```

## Creating Worker Instructions

Create custom instructions for labeling jobs to improve your worker's accuracy in completing their task. Your instructions are accessible when workers select the **Instructions** menu option in the worker UI. Short instructions must be under 255 characters and long instruction must be under 2,048 characters.

There are two kinds of instructions:

- **Short instructions** – These instructions are shown to workers when they select **Instructions** in the worker UI menu. They should provide an easy reference to show the worker the correct way to label an object.
- **Full instructions** – These instructions are shown when workers select **More Instructions** in instructions the pop-up window. We recommend that you provide detailed instructions for completing the task with multiple examples showing edge cases and other difficult situations for labeling objects.

For 3D point cloud and video frame labeling jobs, you can add worker instructions to your label category configuration file. You can use a single string to create instructions or you can add HTML mark up to customize the appearance of your instructions and add images. Make sure that any images you include in your instructions are publicly available, or if your instructions are in Amazon S3, that your workers have read-access so that they can view them.

## Use Input and Output Data

The input data that you provide to Amazon SageMaker Ground Truth is sent to your workers for labeling. You choose the data to send to your workers by creating a single manifest file that defines all of the data that requires labeling or by sending input data objects to an ongoing, streaming labeling job to be labeled in real time.

The output data is the result of your labeling job. The output data file, or *augmented manifest file*, contains label data for each object you send to the labeling job and metadata about the label assigned to data objects.

When you use image classification (single and multi-label), text classification (single and multi-label), object detection, and semantic segmentation built in task types to create a labeling job, you can use the resulting augmented manifest file to launch a SageMaker training job. For a demonstration of how to use an augmented manifest to train an object detection machine learning model with Amazon SageMaker, see [object\\_detection\\_augmented\\_manifest\\_training.ipynb](#). For more information, see [Provide Dataset Metadata to Training Jobs with an Augmented Manifest File \(p. 1864\)](#).

### Topics

- [Input Data \(p. 363\)](#)
- [3D Point Cloud Input Data \(p. 374\)](#)

- [Video Frame Input Data \(p. 398\)](#)
- [Output Data \(p. 404\)](#)

## Input Data

The input data are the data objects that you send to your workforce to be labeled. There are two ways to send data objects to Ground Truth for labeling:

This directory contains one numbered

- Send a list of data objects that require labeling using an input manifest file.
- Send individual data objects in real time to a perpetually running, streaming labeling job.

If you have a dataset that needs to be labeled one time, and you do not require an ongoing labeling job, create a standard labeling job using an input manifest file.

If you want to regularly send new data objects to your labeling job after it has started, create a streaming labeling job. When you create a streaming labeling job, you can optionally use an input manifest file to specify a group of data that you want labeled immediately when the job starts. You can continuously send new data objects to a streaming labeling job as long as it is active.

**Note**

Streaming labeling jobs are only supported through the SageMaker API. You cannot create a streaming labeling job using the SageMaker console.

The following task types have special input data requirements and options:

- For [3D point cloud](#) labeling job input data requirements, see [3D Point Cloud Input Data \(p. 374\)](#).
- For [video frame](#) labeling job input data requirements, see [Video Frame Input Data \(p. 398\)](#).

### Topics

- [Use an Input Manifest File \(p. 363\)](#)
- [Automated Data Setup \(p. 364\)](#)
- [Supported Data Formats \(p. 366\)](#)
- [Ground Truth Streaming Labeling Jobs \(p. 366\)](#)
- [Input Data Quotas \(p. 371\)](#)
- [Filter and Select Data for Labeling \(p. 373\)](#)

## Use an Input Manifest File

Each line in an input manifest file is an entry containing an object, or a reference to an object, to label. An entry can also contain labels from previous jobs and for some task types, additional information.

Input data and the manifest file must be stored in Amazon Simple Storage Service (Amazon S3). Each has specific storage and access requirements, as follows:

- The Amazon S3 bucket that contains the input data must be in the same Amazon Region in which you are running Amazon SageMaker Ground Truth. You must give Amazon SageMaker access to the data stored in the Amazon S3 bucket so that it can read it. For more information about Amazon S3 buckets, see [Working with Amazon S3 buckets](#).
- The manifest file must be in the same Amazon Region as the data files, but it doesn't need to be in the same location as the data files. It can be stored in any Amazon S3 bucket that is accessible to the

Amazon Identity and Access Management (IAM) role that you assigned to Ground Truth when you created the labeling job.

**Note**

3D point cloud and video frame [task types](#) have different input manifest requirements and attributes.

For [3D point cloud task types](#), refer to [Create an Input Manifest File for a 3D Point Cloud Labeling Job \(p. 376\)](#).

For [video frame task types](#), refer to [Create a Video Frame Input Manifest File \(p. 402\)](#).

The manifest is a UTF-8 encoded file in which each line is a complete and valid JSON object. Each line is delimited by a standard line break, \n or \r\n. Because each line must be a valid JSON object, you can't have unescaped line break characters. For more information about data format, see [JSON Lines](#).

Each JSON object in the manifest file can be no larger than 100,000 characters. No single attribute within an object can be larger than 20,000 characters. Attribute names can't begin with \$ (dollar sign).

Each JSON object in the manifest file must contain one of the following keys: `source-ref` or `source`. The value of the keys are interpreted as follows:

- `source-ref` – The source of the object is the Amazon S3 object specified in the value. Use this value when the object is a binary object, such as an image.
- `source` – The source of the object is the value. Use this value when the object is a text value.

The following is an example of a manifest file for files stored in an Amazon S3 bucket:

```
{"source-ref": "S3 bucket location 1"}  
{"source-ref": "S3 bucket location 2"}  
...  
{"source-ref": "S3 bucket location n"}
```

Use the `source-ref` key for image files for bounding box, image classification (single and multi-label), semantic segmentation, and video clips for video classification labeling jobs. 3D point cloud and video frame labeling jobs also use the `source-ref` key but these labeling jobs require additional information in the input manifest file. For more information see [3D Point Cloud Input Data \(p. 374\)](#) and [Video Frame Input Data \(p. 398\)](#).

The following is an example of a manifest file with the input data stored in the manifest:

```
{"source": "Lorem ipsum dolor sit amet"}  
{"source": "consectetur adipiscing elit"}  
...  
{"source": "mollit anim id est laborum"}
```

Use the `source` key for single and multi-label text classification and named entity recognition labeling jobs.

You can include other key-value pairs in the manifest file. These pairs are passed to the output file unchanged. This is useful when you want to pass information between your applications. For more information, see [Output Data \(p. 404\)](#).

## Automated Data Setup

You can use the automated data setup to create manifest files for your labeling jobs in the Ground Truth console using images, videos, video frames, text (.txt) files, and comma-separated value (.csv) files stored

in Amazon S3. When you use automated data setup, you specify an Amazon S3 location where your input data is stored and the input data type, and Ground Truth looks for the files that match that type in the location you specify.

**Note**

Ground Truth does not use an Amazon KMS key to access your input data or write the input manifest file in the Amazon S3 location that you specify. The IAM user or role that creates the labeling job must have permissions to access your input data objects in Amazon S3.

Before using the following procedure, ensure that your input images or files are correctly formatted:

- **Image files** – Image files must comply with the size and resolution limits listed in the tables found in [Input File Size Quota \(p. 371\)](#).
- **Text files** – Text data can be stored in one or more .txt files. Each item that you want labeled must be separated by a standard line break.
- **CSV files** – Text data can be stored in one or more .csv files. Each item that you want labeled must be in a separate row.
- **Videos** – Video files can be any of the following formats: .mp4, .ogg, and .webm. If you want to extract video frames from your video files for object detection or object tracking, see [Provide Video Files \(p. 400\)](#).
- **Video frames** – Video frames are images extracted from a videos. All images extracted from a single video are referred to as a *sequence of video frames*. Each sequence of video frames must have unique prefix keys in Amazon S3. See [Provide Video Frames \(p. 399\)](#). For this data type, see [Automated Video Frame Input Data Setup \(p. 401\)](#)

**Important**

For video frame object detection and video frame object tracking labeling jobs, see [Automated Video Frame Input Data Setup \(p. 401\)](#) to learn how to use the automated data setup.

Use these instructions to automatically set up your input dataset connection with Ground Truth.

**Automatically connect your data in Amazon S3 with Ground Truth**

1. Navigate to the [Create labeling job](#) page in the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.

This link puts you in the North Virginia (us-east-1) Amazon Region. If your input data is in an Amazon S3 bucket in another Region, switch to that Region. To change your Amazon Region, on the [navigation bar](#), choose the name of the currently displayed Region.

2. Select **Create labeling job**.
3. Enter a **Job name**.
4. In the section **Input data setup**, select **Automated data setup**.
5. Enter an Amazon S3 URI for **S3 location for input datasets**.
6. Specify your **S3 location for output datasets**. This is where your output data is stored.
7. Choose your **Data type** using the dropdown list.
8. Use the drop down menu under **IAM Role** to select an execution role. If you select **Create a new role**, specify the Amazon S3 buckets that you want grant this role permission to access. This role must have permission to access the S3 buckets you specified in Steps 5 and 6.
9. Select **Complete data setup**.

The following GIF demonstrates how to use the automated data setup for image data. This example will create a file, `dataset-YYMMDDTHHMMSS.manifest` in the Amazon S3 bucket `example-groundtruth-images` where `YYMMDDTHHMMSS` indicates the year (YY), month (MM), day (DD) and time in hours (HH), minutes (mm) and seconds (ss), that the input manifest file was created.

## Supported Data Formats

When you create an input manifest file for a [built-in task types](#) manually, your input data must be in one of the following support file formats for the respective input data type. To learn about automated data setup, see [Automated Data Setup \(p. 364\)](#).

### Tip

When you use the automated data setup, additional data formats can be used to generate an input manifest file for video frame and text based task types.

Task Types	Input Data Type	Support Formats	Example Input Manifest Line
Bounding Box, Semantic Segmentation, Image Classification (Single Label and Multi-label), Verify and Adjust Labels	Image	.jpg, .jpeg, .png	{ "source-ref": "s3://example-image.png" }
Named Entity Recognition, Text Classification (Single and Multi-Label)	Text	Raw text	{ "source": "Lorem ipsum dolor sit amet" }
Video Classification	Video clips	.mp4, .ogg, and .webm	{ "source-ref": "s3://example-video.mp4" }
Video Frame Object Detection, Video Frame Object Tracking (bounding boxes, polylines, polygons or keypoint)	Video frames and video frame sequence files (for Object Tracking)	<b>Video frames:</b> .jpg, .jpeg, .png <b>Sequence files:</b> .json	Refer to <a href="#">Create a Video Frame Input Manifest File (p. 402)</a> .
3D Point Cloud Semantic Segmentation, 3D Point Cloud Object Detection, 3D Point Cloud Object Tracking	Point clouds and point cloud sequence files (for Object Tracking)	<b>Point clouds:</b> Binary pack format and ASCII. For more information see <a href="#">Accepted Raw 3D Data Formats (p. 375)</a> . <b>Sequence files:</b> .json	Refer to <a href="#">Create an Input Manifest File for a 3D Point Cloud Labeling Job (p. 376)</a> .

## Ground Truth Streaming Labeling Jobs

If you want to perpetually send new data objects to Amazon SageMaker Ground Truth to be labeled, use a streaming labeling job. Streaming labeling jobs allow you to:

- Send new dataset objects to workers in real time using a perpetually running labeling job. Workers continuously receive new data objects to label as long as the labeling job is active and new objects are being sent to it.
- Gain visibility into the number of objects that have been queued and are waiting to be labeled. Use this information to control the flow of data objects sent to your labeling job.
- Receive label data for individual data objects in real time as workers finish labeling them.

Ground Truth streaming labeling jobs remain active until they are manually stopped or have been idle for more than 10 days. You can intermittently send new data objects to workers while the labeling job is active.

If you are a new user of Ground Truth streaming labeling jobs, it is recommended that you review [How It Works \(p. 367\)](#).

Use [Create a Streaming Labeling Job \(p. 343\)](#) to learn how to create a streaming labeling job.

**Note**

Ground Truth streaming labeling jobs are only supported through the SageMaker API.

**Topics**

- [How It Works \(p. 367\)](#)
- [Send Data to a Streaming Labeling Job \(p. 367\)](#)
- [Manage Labeling Requests with an Amazon SQS Queue \(p. 369\)](#)
- [Receive Output Data from a Streaming Labeling Job \(p. 369\)](#)
- [Duplicate Message Handling \(p. 369\)](#)

**How It Works**

When you create a Ground Truth streaming labeling job, the job remains active until it is manually stopped, remains idle for more than 10 days, or is unable to access input data sources. You can intermittently send new data objects to workers while it is active. A worker can continue to receive new data objects in real time as long as the total number of tasks currently available to the worker is less than the value in `MaxConcurrentTaskCount`. Otherwise, the data object is sent to a queue that Ground Truth creates on your behalf in [Amazon Simple Queue Service \(Amazon SQS\)](#) for later processing. These tasks are sent to workers as soon as the total number of tasks currently available to a worker falls below `MaxConcurrentTaskCount`. If a data object is not sent to a worker after 14 days, it expires. You can view the number of tasks pending in the queue and adjust the number of objects you send to the labeling job. For example, you may decrease the speed at which you send objects to the labeling job if the backlog of pending objects moves above a threshold.

**Send Data to a Streaming Labeling Job**

You can optionally submit input data to a streaming labeling job one time when you create the labeling job using an input manifest file. Once the labeling job has started and the state is `InProgress`, you can submit new data objects to your labeling job in real time using your Amazon SNS input topic and Amazon S3 event notifications.

**Submit Data Objects When you Start the Labeling Job (One Time):**

- **Use an Input Manifest File** – You can optionally specify an input manifest file Amazon S3 URI in `ManifestS3Uri` when you create the streaming labeling job. Ground Truth sends each data object in the manifest file to workers for labeling as soon as the labeling job starts. To learn more, see [Create a Manifest File \(Optional\) \(p. 346\)](#).

After you submit a request to create the streaming labeling job, its status will be `Initializing`. Once the labeling job is active, the state changes to `InProgress` and you can start using the real-time options to submit additional data objects for labeling.

**Submit Data Objects in Real Time:**

- **Send data objects using Amazon SNS messages** – You can send Ground Truth new data objects to label by sending an Amazon SNS message. You will send this message to an Amazon SNS input topic that you create and specify when you create your streaming labeling job. For more information, see [Send Data Objects Using Amazon SNS \(p. 368\)](#).

- **Send data objects by placing them in an Amazon S3 bucket** – Each time you add a new data object to an Amazon S3 bucket, you can prompt Ground Truth to process that object for labeling. To do this, you add an event notification to the bucket so that it notifies your Amazon SNS input topic each time a new object is added to (or *created in*) that bucket. For more information, see [Send Data Objects using Amazon S3 \(p. 368\)](#). This option is not available for text-based labeling jobs such as text classification and named entity recognition.

**Important**

If you use the Amazon S3 configuration, do not use the same Amazon S3 location for your input data configuration and your output data. You specify the S3 prefix for your output data when you create a labeling job.

## Send Data Objects Using Amazon SNS

You can send data objects to your streaming labeling job using Amazon Simple Notification Service (Amazon SNS). Amazon SNS is a web service that coordinates and manages the delivery of messages to and from *endpoints* (for example, an email address or Amazon Lambda function). An Amazon SNS *topic* acts as a communication channel between two or more endpoints. You use Amazon SNS to send, or *publish*, new data objects to the topic specified in the `CreateLabelingJob` parameter `SnsTopicArn` in `InputConfig`. The format of these messages is the same as a single line from an [input manifest file](#).

For example, you may send a piece of text to an active text classification labeling job by publishing it to your input topic. The message that you publish may look similar to the following:

```
{"source": "Lorem ipsum dolor sit amet"}
```

To send a new image object to an image classification labeling job, your message may look similar to the following:

```
{"source-ref": "s3://awsexamplebucket/example-image.jpg"}
```

**Note**

You can also include custom deduplication IDs and deduplication keys in your Amazon SNS messages. To learn more, see [Duplicate Message Handling \(p. 369\)](#).

When Ground Truth creates your streaming labeling job, it subscribes to your Amazon SNS input topic.

## Send Data Objects using Amazon S3

You can send one or more new data objects to a streaming labeling job by placing them in an Amazon S3 bucket that is configured with an Amazon SNS event notification. You can set up an event to notify your Amazon SNS input topic anytime a new object is created in your bucket. You must specify this same Amazon SNS input topic in the `CreateLabelingJob` parameter `SnsTopicArn` in `InputConfig`.

Anytime you configure an Amazon S3 bucket to send notifications to Amazon SNS, Ground Truth will publish a test event, "s3:TestEvent", to ensure that the topic exists and that the owner of the Amazon S3 bucket specified has permission to publish to the specified topic. It is recommended that you set up your Amazon S3 connection with Amazon SNS before starting a streaming labeling job. If you do not, this test event may register as a data object and be sent to Ground Truth for labeling.

**Important**

If you use the Amazon S3 configuration, do not use the same Amazon S3 location for your input data configuration and your output data. You specify the S3 prefix for your output data when you create a labeling job.

Once you have configured your Amazon S3 bucket and created your labeling job, you can add objects to your bucket and Ground Truth either sends that object to workers or places it on your Amazon SQS queue.

To learn more, see [Set up Amazon S3 Bucket Event Notifications \(p. 346\)](#).

**Important**

This option is not available for text-based labeling jobs such as text classification and named entity recognition.

### Manage Labeling Requests with an Amazon SQS Queue

When Ground Truth creates your streaming labeling job, it creates an Amazon SQS queue in the Amazon account used to create the labeling job. The queue name is `GroundTruth-labeling_job_name` where `labeling_job_name` is the name of your labeling job, in lowercase letters. When you send data objects to your labeling job, Ground Truth either sends the data objects directly to workers or places the task in your queue to be processed at a later time. If a data object is not sent to a worker after 14 days, it expires and is removed from the queue. You can setup an alarm in Amazon SQS to detect when objects expire and use this mechanism to control the volume of objects you send to your labeling job.

**Important**

Modifying, deleting, or sending objects directly to the Amazon SQS queue associated with your streaming labeling job may lead to job failures.

### Receive Output Data from a Streaming Labeling Job

Your Amazon S3 output bucket is periodically updated with new output data from your streaming labeling job.

Optionally, you can specify an Amazon SNS output topic. Each time a worker submits a labeled object, a notification with the output data is sent to that topic. You can subscribe an endpoint to your SNS output topic to receive notifications or trigger events when you receive output data from a labeling task. Use an Amazon SNS output topic if you want to do real time chaining to another streaming job and receive an Amazon SNS notifications each time a data object is submitted by a worker.

To learn more, see [Subscribe an Endpoint to Your Amazon SNS Output Topic \(p. 345\)](#).

### Duplicate Message Handling

For data objects sent in real time, Ground Truth guarantees idempotency by ensuring each unique object is only sent for labeling once, even if the input message referring to that object is received multiple times (duplicate messages). To do this, each data object sent to a streaming labeling job is assigned a *deduplication ID*, which is identified with a *deduplication key*.

If you send your requests to label data objects directly through your Amazon SNS input topic using Amazon SNS messages, you can optionally choose a custom deduplication key and deduplication IDs for your objects. For more information, see [Specify A Deduplication Key and ID in an Amazon SNS Message \(p. 369\)](#).

If you do not provide your own deduplication key, or if you use the Amazon S3 configuration to send data objects to your labeling job, Ground Truth uses one of the following for the deduplication ID:

- For messages sent directly to your Amazon SNS input topic, Ground Truth uses the SNS message ID.
- For messages that come from an Amazon S3 configuration, Ground Truth creates a deduplication ID by combining the Amazon S3 URL of the object with the `sequencer token` in the message.

### Specify A Deduplication Key and ID in an Amazon SNS Message

When you send a data object to your streaming labeling job using an Amazon SNS message, you have the option to specify your deduplication key and deduplication ID in one of the following ways. In all of these scenarios, identify your deduplication key with `dataset-objectid-attribute-name`.

#### Bring Your Own Deduplication Key and ID

Create your own deduplication key and deduplication ID by configuring your Amazon SNS message as follows. Replace `byo-key` with your key and `UniqueId` with the deduplication ID for that data object.

```
{
    "source-ref": "s3://bucket/prefix/object1",
    "dataset-objectid-attribute-name": "byo-key",
    "byo-key": "UniqueId"
}
```

Your deduplication key can be up to 140 characters. Supported patterns include: `^[ $a-zA-Z0-9 ](-*[ a-zA-Z0-9 ])*$`.

Your deduplication ID can be up to 1,024 characters. Supported patterns include: `^(https|s3)://([ ^/]+)?(.*$)`.

### Use an Existing Key for your Deduplication Key

You can use an existing key in your message as the deduplication key. When you do this, the value associated with that key is used for the deduplication ID.

For example, you can specify use the `source-ref` key as your deduplication key by formatting your message as follows:

```
{
    "source-ref": "s3://bucket/prefix/object1",
    "dataset-objectid-attribute-name": "source-ref"
}
```

In this example, Ground Truth uses `s3://bucket/prefix/object1` for the deduplication id.

### Find Deduplication Key and ID in Your Output Data

You can see the deduplication key and ID in your output data. The deduplication key is identified by `dataset-objectid-attribute-name`.

When you use your own custom deduplication key, your output contains something similar to the following:

```
"dataset-objectid-attribute-name": "byo-key",
"byo-key": "UniqueId",
```

When you do not specify a key, you can find the deduplication ID that Ground Truth assigned to your data object as follows. The `$label-attribute-name-object-id` parameter identifies your deduplication ID.

```
{
    "source-ref": "s3://bucket/prefix/object1",
    "dataset-objectid-attribute-name": "$label-attribute-name-object-id"
    "$label-attribute-name": 0,
    "$label-attribute-name-metadata": {...},
    "$label-attribute-name-object-id": "<service-generated-key>"
}
```

For `<service-generated-key>`, if the data object came through an Amazon S3 configuration, Ground Truth adds a unique value used by the service and emits a new field keyed by `$sequencer` which shows the Amazon S3 sequencer used. If object was fed to SNS directly, Ground Truth use the SNS message ID.

**Note**

Do not use the \$ character in your label attribute name.

## Input Data Quotas

Input datasets used in semantic segmentation labeling jobs have a quota of 20,000 items. For all other labeling job types, the dataset size quota is 100,000 items. To request an increase to the quota for labeling jobs other than semantic segmentation jobs, review the procedures in [Amazon Service Quotas](#) to request a quota increase.

Input image data for active and non-active learning labeling jobs must not exceed size and resolution quotas. *Active learning* refers to labeling job that use [automated data labeling](#). *Non-active learning* refers to labeling jobs that don't use automated data labeling.

Additional quotas apply for label categories for all task types, and for input data and labeling category attributes for 3D point cloud and video frame task types.

### Input File Size Quota

Input files can't exceed the following size- quotas for both active and non-active learning labeling jobs. There is no input file size quota for videos used in [video classification](#) labeling jobs.

Labeling Job Task Type	Input File Size Quota
Image classification	40 MB
Bounding box (Object detection)	40 MB
Semantic segmentation	40 MB
Bounding box (Object detection) label adjustment	40 MB
Semantic segmentation label adjustment	40 MB
Bounding box (Object detection) label verification	40 MB
Semantic segmentation label verification	40 MB

### Input Image Resolution Quotas

Image file resolution refers to the number of pixels in an image, and determines the amount of detail an image holds. Image resolution quotas differ depending on the labeling job type and the SageMaker built-in algorithm used. The following table lists the resolution quotas for images used in active and non-active learning labeling jobs.

Labeling Job Task Type	Resolution Quota - Non Active Learning	Resolution Quota - Active Learning
Image classification	100 million pixels	3840 x 2160 pixels (4 K)
Bounding box (Object detection)	100 million pixels	3840 x 2160 pixels (4 K)
Semantic segmentation	100 million pixels	1920 x 1080 pixels (1080 p)
Object detection label adjustment	100 million pixels	3840 x 2160 pixels (4 K)

Labeling Job Task Type	Resolution Quota - Non Active Learning	Resolution Quota - Active Learning
Semantic segmentation label adjustment	100 million pixels	1920 x 1080 pixels (1080 p)
Object detection label verification	100 million pixels	Not available
Semantic segmentation label verification	100 million pixels	Not available

### Label Category Quotas

Each labeling job task type has a quota for the number of label categories you can specify. Workers select label categories to create annotations. For example, you may specify label categories *car*, *pedestrian*, and *biker* when creating a bounding box labeling job and workers will select the *car* category before drawing bounding boxes around cars.

**Important**

Label category names cannot exceed 256 characters.

All label categories must be unique. You cannot specify duplicate label categories.

The following label category limits apply to labeling jobs. Quotas for label categories depend on whether you use the SageMaker API operation `CreateLabelingJob` or the console to create a labeling job.

Labeling Job Task Type	Label Category Quota - API	Label Category Quota - Console
Image classification (Multi-label)	50	50
Image classification (Single label)	Unlimited	30
Bounding box (Object detection)	50	50
Label verification	Unlimited	30
Semantic segmentation (with active learning)	20	10
Semantic segmentation (without active learning)	Unlimited	10
Named entity recognition	Unlimited	30
Text classification (Multi-label)	50	50
Text classification (Single label)	Unlimited	30
Video classification	30	30
Video frame object detection	30	30
Video frame object tracking	30	30
3D point cloud object detection	30	30
3D point cloud object tracking	30	30

Labeling Job Task Type	Label Category Quota - API	Label Category Quota - Console
3D point cloud semantic segmentation	30	30

### 3D Point Cloud and Video Frame Labeling Job Quotas

The following quotas apply for 3D point cloud and video frame labeling job input data.

Labeling Job Task Type	Input Data Quota
Video frame object detection	2,000 video frames (images) per sequence
Video frame object detection	10 video frame sequences per manifest file
Video frame object tracking	2,000 video frames (images) per sequence
Video frame object tracking	10 video frame sequences per manifest file
3D point cloud object detection	100,000 point cloud frames per labeling job
3D point cloud object tracking	100,000 point cloud frame sequences per labeling job
3D point cloud object tracking	500 point cloud frames in each sequence file

When you create a video frame or 3D point cloud labeling job, you can add one or more *label category attributes* to each label category that you specify to have workers provide more information about an annotation.

Each label category attribute has a single label category attribute name, and a list of one or more options (values) to choose from. To learn more, see [Worker User Interface \(UI\) \(p. 260\)](#) for 3D point cloud labeling jobs and [Worker User Interface \(UI\) \(p. 212\)](#) for video frame labeling jobs.

The following quotas apply to the number of label category attributes names and values you can specify for labeling jobs.

Labeling Job Task Type	Label Category Attribute (name) Quota	Label Category Attribute Values Quota
Video frame object detection	10	10
Video frame object tracking	10	10
3D point cloud object detection	10	10
3D point cloud object tracking	10	10
3D point cloud semantic segmentation	10	10

### Filter and Select Data for Labeling

You can use the Amazon SageMaker console to select a portion of your dataset for labeling. The data must be stored in an Amazon S3 bucket. You have three options:

- Use the full dataset.
- Choose a randomly selected sample of the dataset.
- Specify a subset of the dataset using a query.

The following options are available in the **Labeling jobs** section of the [SageMaker console](#) after selecting **Create labeling job**. To learn how to create a labeling job in the console, see [Getting started \(p. 166\)](#). To configure the dataset that you use for labeling, in the **Job overview** section, choose **Additional configuration**.

### Use the Full Dataset

When you choose to use the **Full dataset**, you must provide a manifest file for your data objects. You can provide the path of the Amazon S3 bucket that contains the manifest file or use the SageMaker console to create the file. To learn how to create a manifest file using the console, see [Automated Data Setup \(p. 364\)](#).

### Choose a Random Sample

When you want to label a random subset of your data, select **Random sample**. The dataset is stored in the Amazon S3 bucket specified in the **Input dataset location** field.

After you have specified the percentage of data objects that you want to include in the sample, choose **Create subset**. SageMaker randomly picks the data objects for your labeling job. After the objects are selected, choose **Use this subset**.

SageMaker creates a manifest file for the selected data objects. It also modifies the value in the **Input dataset location** field to point to the new manifest file.

### Specify a Subset

You can specify a subset of your data objects using an Amazon S3 `SELECT` query on the object file names.

The `SELECT` statement of the SQL query is defined for you. You provide the `WHERE` clause to specify which data objects should be returned.

For more information about the Amazon S3 `SELECT` statement, see [Selecting Content from Objects](#).

Choose **Create subset** to start the selection, and then choose **Use this subset** to use the selected data.

SageMaker creates a manifest file for the selected data objects. It also updates the value in the **Input dataset location** field to point to the new manifest file.

## 3D Point Cloud Input Data

To create a 3D point cloud labeling job, you must create an input manifest file. Use this topic to learn the formatting requirements of the input manifest file for each task type. To learn about the raw input data formats Ground Truth accepts for 3D point cloud labeling jobs, see the section [Accepted Raw 3D Data Formats \(p. 375\)](#).

Use your [labeling job task type](#) to choose a topics on [Create an Input Manifest File for a 3D Point Cloud Labeling Job \(p. 376\)](#) to learn about the formatting requirements for each line of your input manifest file.

### Topics

- [Accepted Raw 3D Data Formats \(p. 375\)](#)
- [Create an Input Manifest File for a 3D Point Cloud Labeling Job \(p. 376\)](#)
- [Understand Coordinate Systems and Sensor Fusion \(p. 389\)](#)

## Accepted Raw 3D Data Formats

Ground Truth uses your 3D point cloud data to render a 3D scenes that workers annotate. This section describes the raw data formats that are accepted for point cloud data and sensor fusion data for a point cloud frame. To learn how to create an input manifest file to connect your raw input data files with Ground Truth, see [Create an Input Manifest File for a 3D Point Cloud Labeling Job \(p. 376\)](#).

For each frame, Ground Truth supports Compact Binary Pack Format (.bin) and ASCII (.txt) files. These files contain information about the location (x, y, and z coordinates) of all points that make up that frame, and, optionally, information about the pixel color of each point for colored point clouds. When you create a 3D point cloud labeling job input manifest file, you can specify the format of your raw data in the `format` parameter.

The following table lists elements that Ground Truth supports in point cloud frame files to describe individual points.

Symbol	Value
x	The x coordinate of the point.
y	The y coordinate of the point.
z	The z coordinate of the point.
i	The intensity of the point.
r	The red color channel component. An 8-bit value (0-255).
g	The green color channel component. An 8-bit value (0-255)
b	The blue color channel component. An 8-bit value (0-255)

Ground Truth assumes the following about your input data:

- All of the positional coordinates (x, y, z) are in meters.
- All the pose headings (qx, qy, qz, qw) are measured in Spatial [Quaternions](#) .

### Compact Binary Pack Format

The Compact Binary Pack Format represents a point cloud as an ordered set of a stream of points. Each point in the stream is an ordered binary pack of 4-byte float values in some variant of the form `xyzirgb`. The x, y, and z elements are required and additional information about that pixel can be included in a variety of ways using i, r, g, and b.

To use a binary file to input point cloud frame data to a Ground Truth 3D point cloud labeling job, enter `binary/` in the `format` parameter for your input manifest file and replace `/` with the order of elements in each binary pack. For example, you may enter one of the following for the `format` parameter.

- `binary/xyz` – When you use this format, your point element stream would be in the following order: `x1y1z1i1x2y2z2i2...`
- `binary/xyzrgb` – When you use this format, your point element stream would be in the following order: `x1y1z1r1g1b1x2y2z2r2g2b2...`
- `binary/xyzirgb` – When you use this format, your point element stream would be in the following order: `x1y1z1i1r1g1b1x2y2z2i2r2g2b2...`

When you use a binary file for your point cloud frame data, if you do not enter a value for `format`, the default pack format `binary/xyzzi` is used.

### ASCII Format

The ASCII format uses a text file to represent a point cloud, where each line in the ASCII point cloud file represents a single point. Each point is a line in the text file and contains white space separated values, each of which is a 4-byte float ASCII values. The `x`, `y`, and `z` elements are required for each point and additional information about that point can be included in a variety of ways using `i`, `r`, `g`, and `b`.

To use a text file to input point cloud frame data to a Ground Truth 3D point cloud labeling job, enter `text/` in the `format` parameter for your input manifest file and replace `text/` with the order of point elements on each line.

For example, if you enter `text/xyzzi` for `format`, your text file for each point cloud frame should look similar to the following:

```
x1 y1 z1 i1
x2 y2 z2 i2
...
...
```

If you enter `text/xyzrgb`, your text file should look similar to the following:

```
x1 y1 z1 r1 g1 b1
x2 y2 z2 r2 g2 b1
...
...
```

When you use a text file for your point cloud frame data, if you do not enter a value for `format`, the default format `text/xyzzi` will be used.

### Point Cloud Resolution Limits

Ground Truth does not have a resolution limit for 3D point cloud frames. However, we recommend that you limit each point cloud frame to 500K points for optimal performance. When Ground Truth renders the 3D point cloud visualization, it must be viewable on your workers' computers, which depends on workers' computer hardware. Point cloud frames that are larger than 1 million points may not render on standard machines, or may take too long to load.

## Create an Input Manifest File for a 3D Point Cloud Labeling Job

When you create a labeling job, you provide an input manifest file where each line of the manifest describes a unit of task to be completed by annotators. The format of your input manifest file depends on your task type.

- If you are creating a 3D point cloud **object detection** or **semantic segmentation** labeling job, each line in your input manifest file contains information about a single 3D point cloud frame. This is called a *point cloud frame input manifest*. To learn more, see [Create a Point Cloud Frame Input Manifest File \(p. 376\)](#).
- If you are creating a 3D point cloud **object tracking** labeling job, each line of your input manifest file contains a sequence of 3D point cloud frames and associated data. This is called a *point cloud sequence input manifest*. To learn more, see [Create a Point Cloud Sequence Input Manifest \(p. 383\)](#).

### Create a Point Cloud Frame Input Manifest File

The manifest is a UTF-8 encoded file in which each line is a complete and valid JSON object. Each line is delimited by a standard line break, `\n` or `\r\n`. Because each line must be a valid JSON object, you can't

have unescaped line break characters. In the single-frame input manifest file, each line in the manifest contains data for a single point cloud frame. The point cloud frame data can either be stored in binary or ASCII format (see [Accepted Raw 3D Data Formats \(p. 375\)](#)). This is the manifest file formatting required for 3D point cloud object detection and semantic segmentation. Optionally, you can also provide camera sensor fusion data for each point cloud frame.

Ground Truth supports point cloud and video camera sensor fusion in the [world coordinate system \(p. 389\)](#) for all modalities. If you can obtain your 3D sensor extrinsic (like a LiDAR extrinsic), we recommend that you transform 3D point cloud frames into the world coordinate system using the extrinsic. For more information, see [Sensor Fusion \(p. 391\)](#).

However, if you cannot obtain a point cloud in world coordinate system, you can provide coordinates in the original coordinate system that the data was captured in. If you are providing camera data for sensor fusion, it is recommended that you provide LiDAR sensor and camera pose in the world coordinate system.

To create a single-frame input manifest file, you will identify the location of each point cloud frame that you want workers to label using the `source-ref` key. Additionally, you must use the `source-ref-metadata` key to identify the format of your dataset, a timestamp for that frame, and, optionally, sensor fusion data and video camera images.

The following example demonstrates the syntax used for an input manifest file for a single-frame point cloud labeling job. The example includes two point cloud frames. For details about each parameter, see the table following this example.

**Important**

Each line in your input manifest file must be in [JSON Lines](#) format. The following code block shows an input manifest file with two JSON objects. Each JSON object is used to point to and provide details about a single point cloud frame. The JSON objects have been expanded for readability, but you must minimize each JSON object to fit on a single line when creating an input manifest file. An example is provided under this code block.

```
{  
    "source-ref": "s3://awsexamplebucket/examplefolder/frame1.bin",  
    "source-ref-metadata": {  
        "format": "binary/xyzzi",  
        "unix-timestamp": 1566861644.759115,  
        "ego-vehicle-pose": {  
            "position": {  
                "x": -2.7161461413869947,  
                "y": 116.25822288149078,  
                "z": 1.8348751887989483  
            },  
            "heading": {  
                "qx": -0.02111296123795955,  
                "qy": -0.006495469416730261,  
                "qz": -0.008024565904865688,  
                "qw": 0.9997181192298087  
            }  
        },  
        "prefix": "s3://awsexamplebucket/lidar_singleframe_dataset/someprefix/",  
        "images": [  
            {  
                "image-path": "images/frame300.bin_camera0.jpg",  
                "unix-timestamp": 1566861644.759115,  
                "fx": 847.7962624528487,  
                "fy": 850.0340893791985,  
                "cx": 576.2129134707038,  
                "cy": 317.2423573573745,  
                "k1": 0,  
                "k2": 0,  
                "k3": 0,  
                "k4": 0  
            }  
        ]  
    }  
}
```

```

        "k4": 0,
        "p1": 0,
        "p2": 0,
        "skew": 0,
        "position": {
            "x": -2.2722515189268138,
            "y": 116.86003310568965,
            "z": 1.454614668542299
        },
        "heading": {
            "qx": 0.7594754093069037,
            "qy": 0.02181790885672969,
            "qz": -0.02461725233103356,
            "qw": -0.6496916273040025
        },
        "camera-model": "pinhole"
    }
}
{
    "source-ref": "s3://awsexamplebucket/examplefolder/frame2.bin",
    "source-ref-metadata": {
        "format": "binary/xyzzi",
        "unix-timestamp": 1566861632.759133,
        "ego-vehicle-pose": {
            "position": {
                "x": -2.7161461413869947,
                "y": 116.25822288149078,
                "z": 1.8348751887989483
            },
            "heading": {
                "qx": -0.02111296123795955,
                "qy": -0.006495469416730261,
                "qz": -0.008024565904865688,
                "qw": 0.9997181192298087
            }
        },
        "prefix": "s3://awsexamplebucket/lidar_singleframe_dataset/someprefix/",
        "images": [
            {
                "image-path": "images/frame300.bin_camera0.jpg",
                "unix-timestamp": 1566861644.759115,
                "fx": 847.7962624528487,
                "fy": 850.0340893791985,
                "cx": 576.2129134707038,
                "cy": 317.2423573573745,
                "k1": 0,
                "k2": 0,
                "k3": 0,
                "k4": 0,
                "p1": 0,
                "p2": 0,
                "skew": 0,
                "position": {
                    "x": -2.2722515189268138,
                    "y": 116.86003310568965,
                    "z": 1.454614668542299
                },
                "heading": {
                    "qx": 0.7594754093069037,
                    "qy": 0.02181790885672969,
                    "qz": -0.02461725233103356,
                    "qw": -0.6496916273040025
                },
                "camera-model": "pinhole"
            }
        ]
}

```

```
    }
}
```

When you create an input manifest file, you must collapse your JSON objects to fit on a single line. For example, the code block above would appear as follows in an input manifest file:

```
{"source-ref":"s3://awsexamplebucket/examplefolder/frame1.bin","source-ref-metadata": {"format":"binary/xyz", "unix-timestamp":1566861644.759115, "ego-vehicle-pose":{"position": {"x":-2.7161461413869947, "y":116.25822288149078, "z":1.8348751887989483}, "heading": {"qx":-0.02111296123795955, "qy":-0.006495469416730261, "qz":-0.008024565904865688, "qw":0.999718119229808 awsexamplebucket/lidar_singleframe_dataset/someprefix/", "images": [{"image-path": "images/frame300.bin_camera0.jpg", "unix- timestamp":1566861644.759115, "fx":847.7962624528487, "fy":850.0340893791985, "cx":576.2129134707038, "cy": { "x":-2.2722515189268138, "y":116.86003310568965, "z":1.454614668542299}, "heading": {"qx":0.7594754093069037, "qy":0.02181790885672969, "qz":-0.02461725233103356, "qw":-0.6496916273040025}, "model": "pinhole"}]}}, {"source-ref":"s3://awsexamplebucket/examplefolder/frame2.bin","source-ref-metadata": {"format":"binary/xyz", "unix-timestamp":1566861632.759133, "ego-vehicle-pose":{"position": {"x":-2.7161461413869947, "y":116.25822288149078, "z":1.8348751887989483}, "heading": {"qx":-0.02111296123795955, "qy":-0.006495469416730261, "qz":-0.008024565904865688, "qw":0.999718119229808 awsexamplebucket/lidar_singleframe_dataset/someprefix/", "images": [{"image-path": "images/frame300.bin_camera0.jpg", "unix- timestamp":1566861644.759115, "fx":847.7962624528487, "fy":850.0340893791985, "cx":576.2129134707038, "cy": { "x":-2.2722515189268138, "y":116.86003310568965, "z":1.454614668542299}, "heading": {"qx":0.7594754093069037, "qy":0.02181790885672969, "qz":-0.02461725233103356, "qw":-0.6496916273040025}, "model": "pinhole"}]}}}
```

The following table shows the parameters you can include in your input manifest file:

Parameter	Required	Accepted Values	Description
source-ref	Yes	String <b>Accepted string value format:</b> <code>s3://&lt;bucket-name&gt;/&lt;folder-name&gt;/point-cloud-frame-file</code>	The Amazon S3 location of a single point cloud frame.
source-ref-metadata	Yes	JSON object <b>Accepted parameters:</b> <code>format, unix-timestamp, ego-vehicle-pose, position, prefix, images</code>	Use this parameter to include additional information about the point cloud in source-ref, and to provide camera data for sensor fusion.
format	No	String <b>Accepted string values:</b> "binary/xyz", "binary/xyz", "binary/xyz", "binary/xyzrgb", "binary/xyzrgb", "text/xyz", "text/xyz",	Use this parameter to specify the format of your point cloud data. For more information, see <a href="#">Accepted Raw 3D Data Formats (p. 375)</a> .

Parameter	Required	Accepted Values	Description
		<p>"text/xyzrgb", "text/xyzirgb"</p> <p><b>Default Values:</b></p> <p>When the file identified in <code>source-ref</code> has a .bin extension, <code>binary/xyz</code></p> <p>When the file identified in <code>source-ref</code> has a .txt extension, <code>text/xyz</code></p>	
<code>unix-timestamp</code>	Yes	<p>Number</p> <p>A unix timestamp.</p>	The unix timestamp is the number of seconds since January 1st, 1970 until the UTC time that the data was collected by a sensor.
<code>ego-vehicle-pose</code>	No	JSON object	The pose of the device used to collect the point cloud data. For more information about this parameter, see <a href="#">Include Vehicle Pose Information in Your Input Manifest (p. 381)</a> .
<code>prefix</code>	No	<p>String</p> <p><b>Accepted string value format:</b></p> <p>s3://&lt;bucket-name&gt;/&lt;folder-name&gt;/</p>	<p>The location in Amazon S3 where your metadata, such as camera images, is stored for this frame.</p> <p>The prefix must end with a forward slash: /.</p>
<code>images</code>	No	List	A list of parameters describing color camera images used for sensor fusion. You can include up to 8 images in this list. For more information about the parameters required for each image, see <a href="#">Include Camera Data in Your Input Manifest (p. 381)</a> .

## Include Vehicle Pose Information in Your Input Manifest

Use the ego-vehicle location to provide information about the location of the vehicle used to capture point cloud data. Ground Truth uses this information to compute LiDAR extrinsic matrix.

Ground Truth uses extrinsic matrices to project labels to and from the 3D scene and 2D images. For more information, see [Sensor Fusion \(p. 391\)](#).

The following table provides more information about the position and orientation (heading) parameters that are required when you provide ego-vehicle information.

Parameter	Required	Accepted Values	Description
position	Yes	JSON object  <b>Required Parameters:</b> x, y, and z. Enter numbers for these parameters.	The translation vector of the ego vehicle in the world coordinate system.
heading	Yes	JSON Object  <b>Required Parameters:</b> qx, qy, qz, and qw. Enter numbers for these parameters.	The orientation of the frame of reference of the device or sensor mounted on the vehicle sensing the surrounding, measured in <a href="#">quaternions</a> , (qx, qy, qz, qw) in the a coordinate system.

## Include Camera Data in Your Input Manifest

If you want to include video camera data with a frame, use the following parameters to provide information about each image. The **Required** column below applies when the `images` parameter is included in the input manifest file under `source-ref-metadata`. You are not required to include `images` in your input manifest file.

If you include camera images, you must include information about the camera position and heading used the capture the images in the world coordinate system.

If your images are distorted, Ground Truth can automatically undistort them using information you provide about the image in your input manifest file, including distortion coefficients (`k1`, `k2`, `k3`, `k4`, `p1`, `p1`), the camera model and the camera intrinsic matrix. The intrinsic matrix is made up of focal length (`fx`, ), and the principal point (`cx`, `cy`). See [Intrinsic Matrix \(p. 393\)](#) to learn how Ground Truth uses the camera intrinsic. If distortion coefficients are not included, Ground Truth will not undistort an image.

Parameter	Required	Accepted Values	Description
image-path	Yes	String  <b>Example of format:</b> <code>&lt;folder-name&gt;/&lt;imagefile.png&gt;</code>	The relative location, in Amazon S3 of your image file. This relative path will be appended to the path you specify in <code>prefix</code> .

Parameter	Required	Accepted Values	Description
unix-timestamp	Yes	Number	The unix timestamp is the number of seconds since January 1st, 1970 until the UTC time that the data was collected by a camera.
camera-model	No	<p>String:</p> <p><b>Accepted Values:</b> "pinhole", "fisheye"</p> <p><b>Default:</b> "pinhole"</p>	The model of the camera used to capture the image. This information is used to undistort camera images.
fx, fy	Yes	Numbers	The focal length of the camera, in the x (fx) and y (fy) directions.
cx, cy	Yes	Numbers	The x (cx) and y (cy) coordinates of the principal point.
k1, k2, k3, k4	No	Number	Radial distortion coefficients. Supported for both <b>fisheye</b> and <b>pinhole</b> camera models.
p1, p2	No	Number	Tangential distortion coefficients. Supported for <b>pinhole</b> camera models.
skew	No	Number	A parameter to measure the skew of an image.
position	Yes	<p>JSON object</p> <p><b>Required Parameters:</b> x, y, and z. Enter numbers for these parameters.</p>	The location or origin of the frame of reference of the camera mounted on the vehicle capturing images.
heading	Yes	<p>JSON Object</p> <p><b>Required Parameters:</b> qx, qy, qz, and qw. Enter numbers for these parameters.</p>	The orientation of the frame of reference of the camera mounted on the vehicle capturing images, measured using <b>quaternions</b> , (qx, qy, qz, qw), in the world coordinate system.

## Point Cloud Frame Limits

You can include up to 100,000 point cloud frames in your input manifest file. 3D point cloud labeling job have longer pre-processing times than other Ground Truth task types. For more information, see [Job Pre-processing Time \(p. 259\)](#).

## Create a Point Cloud Sequence Input Manifest

The manifest is a UTF-8 encoded file in which each line is a complete and valid JSON object. Each line is delimited by a standard line break, \n or \r\n. Because each line must be a valid JSON object, you can't have unescaped line break characters. In the point cloud sequence input manifest file, each line in the manifest contains a sequence of point cloud frames. The point cloud data for each frame in the sequence can either be stored in binary or ASCII format. For more information, see [Accepted Raw 3D Data Formats \(p. 375\)](#). This is the manifest file formatting required for 3D point cloud object tracking. Optionally, you can also provide point attribute and camera sensor fusion data for each point cloud frame. When you create a sequence input manifest file, you must provide LiDAR and video camera sensor fusion data in a [world coordinate system \(p. 389\)](#).

The following example demonstrates the syntax used for an input manifest file when each line in the manifest is a sequence file. Each line in your input manifest file must be in [JSON Lines](#) format.

```
{"source-ref": "s3://awsexamplebucket/example-folder/seq1.json"}  
{"source-ref": "s3://awsexamplebucket/example-folder/seq2.json"}
```

The data for each sequence of point cloud frames needs to be stored in a JSON data object. The following is an example of the format you use for a sequence file. Information about each frame is included as a JSON object and is listed in the `frames` list. This is an example of a sequence file with two point cloud frame files, `frame300.bin` and `frame303.bin`. The `...` is used to indicated where you should include information for additional frames. Add a JSON object for each frame in the sequence.

The following code block includes a JSON object for a single sequence file. The JSON object has been expanded for readability.

```
{
  "seq-no": 1,
  "prefix": "s3://awsexamplebucket/example_lidar_sequence_dataset/seq1/",
  "number-of-frames": 100,
  "frames": [
    {
      "frame-no": 300,
      "unix-timestamp": 1566861644.759115,
      "frame": "example_lidar_frames/frame300.bin",
      "format": "binary/xyz",
      "ego-vehicle-pose": {
        "position": {
          "x": -2.7161461413869947,
          "y": 116.25822288149078,
          "z": 1.8348751887989483
        },
        "heading": {
          "qx": -0.02111296123795955,
          "qy": -0.006495469416730261,
          "qz": -0.008024565904865688,
          "qw": 0.9997181192298087
        }
      },
      "images": [
        {
          "image-path": "example_images/frame300.bin_camera0.jpg",
          "unix-timestamp": 1566861644.759115,
          "fx": 847.7962624528487,
          "fy": 500.0,
          "cx": 320.0,
          "cy": 240.0
        }
      ]
    }
  ]
}
```

```

    "fy": 850.0340893791985,
    "cx": 576.2129134707038,
    "cy": 317.2423573573745,
    "k1": 0,
    "k2": 0,
    "k3": 0,
    "k4": 0,
    "p1": 0,
    "p2": 0,
    "skew": 0,
    "position": {
        "x": -2.2722515189268138,
        "y": 116.86003310568965,
        "z": 1.454614668542299
    },
    "heading": {
        "qx": 0.7594754093069037,
        "qy": 0.02181790885672969,
        "qz": -0.02461725233103356,
        "qw": -0.6496916273040025
    },
    "camera-model": "pinhole"
}
],
{
    "frame-no": 303,
    "unix-timestamp": 1566861644.759115,
    "frame": "example_lidar_frames/frame303.bin",
    "format": "text/xyzzi",
    "ego-vehicle-pose": {...},
    "images": [...]
},
...
]
}

```

The following table provides details about the top-level parameters of a sequence file. For detailed information about the parameters required for individual frames in the sequence file, see [Parameters for Individual Point Cloud Frames \(p. 385\)](#).

Parameter	Required	Accepted Values	Description
seq-no	Yes	Integer	The ordered number of the sequence.
prefix	Yes	String <b>Accepted Values:</b> s3://<bucket-name>/<prefix>/	The Amazon S3 location where the sequence files are located. The prefix must end with a forward slash: /.
number-of-frames	Yes	Integer	The total number of frames included in the sequence file. This number must match the total number of frames listed in the <code>frames</code> parameter in the next row.

Parameter	Required	Accepted Values	Description
frames	Yes	List of JSON objects	<p>A list of frame data. The length of the list must equal <code>number-of-frames</code>. In the worker UI, frames in a sequence will be the same as the order of frames in this array.</p> <p>For details about the format of each frame, see <a href="#">Parameters for Individual Point Cloud Frames (p. 385)</a>.</p>

### Parameters for Individual Point Cloud Frames

The following table shows the parameters you can include in your input manifest file.

Parameter	Required	Accepted Values	Description
frame-no	No	Integer	<p>A frame number. This is an optional identifier specified by the customer to identify the frame within a sequence. It is not used by Ground Truth.</p>
unix-timestamp	Yes	Number	<p>The unix timestamp is the number of seconds since January 1st, 1970 until the UTC time that the data was collected by a sensor.</p> <p>The timestamp for each frame must be different and timestamps must be sequential because they are used for cuboid interpolation. Ideally, this should be the real timestamp when the data was collected. If this is not available, you must use an incremental sequence of timestamps, where the first frame in your sequence file corresponds to the</p>

Parameter	Required	Accepted Values	Description
			first timestamp in the sequence.
frame	Yes	<p>String</p> <p><b>Example of format</b></p> <p style="color: red;"><i>&lt;folder-name&gt;/&lt;sequence-file.json&gt;</i></p>	The relative location, in Amazon S3 of your sequence file. This relative path will be appended to the path you specify in <code>prefix</code> .
format	No	<p>String</p> <p><b>Accepted string values:</b> "binary/xyz", "binary/xyzi", "binary/xyzrgb", "binary/xyzirgb", "text/xyz", "text/xyzi", "text/xyzrgb", "text/xyzirgb"</p> <p><b>Default Values:</b></p> <p>When the file identified in <code>source-ref</code> has a .bin extension, binary/xyzi</p> <p>When the file identified in <code>source-ref</code> has a .txt extension, text/xyzi</p>	Use this parameter to specify the format of your point cloud data. For more information, see <a href="#">Accepted Raw 3D Data Formats (p. 375)</a> .
ego-vehicle-pose	No	JSON object	The pose of the device used to collect the point cloud data. For more information about this parameter, see <a href="#">Include Vehicle Pose Information in Your Input Manifest (p. 387)</a> .
prefix	No	<p>String</p> <p><b>Accepted string value format:</b></p> <p style="color: red;"><i>s3://&lt;bucket-name&gt;/&lt;folder-name&gt;/</i></p>	<p>The location in Amazon S3 where your metadata, such as camera images, is stored for this frame.</p> <p>The prefix must end with a forward slash: /.</p>

Parameter	Required	Accepted Values	Description
images	No	List	A list parameters describing color camera images used for sensor fusion. You can include up to 8 images in this list. For more information about the parameters required for each image, see <a href="#">Include Camera Data in Your Input Manifest (p. 387)</a> .

### Include Vehicle Pose Information in Your Input Manifest

Use the ego-vehicle location to provide information about the pose of the vehicle used to capture point cloud data. Ground Truth use this information to compute LiDAR extrinsic matrices.

Ground Truth uses extrinsic matrices to project labels to and from the 3D scene and 2D images. For more information, see [Sensor Fusion \(p. 391\)](#).

The following table provides more information about the position and orientation (heading) parameters that are required when you provide ego-vehicle information.

Parameter	Required	Accepted Values	Description
position	Yes	JSON object  <b>Required Parameters:</b> x, y, and z. Enter numbers for these parameters.	The translation vector of the ego vehicle in the world coordinate system.
heading	Yes	JSON Object  <b>Required Parameters:</b> qx, qy, qz, and qw. Enter numbers for these parameters.	The orientation of the frame of reference of the device or sensor mounted on the vehicle sensing the surrounding, measured in <a href="#">quaternions</a> , (qx, qy, qz, qw) in the a coordinate system.

### Include Camera Data in Your Input Manifest

If you want to include color camera data with a frame, use the following parameters to provide information about each image. The **Required** column in the following table applies when the `images` parameter is included in the input manifest file. You are not required to include images in your input manifest file.

If you include camera images, you must include information about the position and orientation (heading) of the camera used the capture the images.

If your images are distorted, Ground Truth can automatically undistort them using information you provide about the image in your input manifest file, including distortion coefficients ( $k_1, k_2, k_3, k_4, p_1, p_2$ ), camera model and focal length ( $f_x, f_y$ ), and the principal point ( $c_x, c_y$ ). To learn more about these coefficients and undistorting images, see [Camera calibration With OpenCV](#). If distortion coefficients are not included, Ground Truth will not undistort an image.

Parameter	Required	Accepted Values	Description
image-path	Yes	String <b>Example of format:</b> <i>&lt;folder-name&gt;/&lt;imagefile.png&gt;</i>	The relative location, in Amazon S3 of your image file. This relative path will be appended to the path you specify in <code>prefix</code> .
unix-timestamp	Yes	Number	The timestamp of the image.
camera-model	No	String: <b>Accepted Values:</b> "pinhole", "fisheye" <b>Default:</b> "pinhole"	The model of the camera used to capture the image. This information is used to undistort camera images.
$f_x, f_y$	Yes	Numbers	The focal length of the camera, in the x ( $f_x$ ) and y ( $f_y$ ) directions.
$c_x, c_y$	Yes	Numbers	The x ( $c_x$ ) and y ( $c_y$ ) coordinates of the principal point.
$k_1, k_2, k_3, k_4$	No	Number	Radial distortion coefficients. Supported for both <b>fisheye</b> and <b>pinhole</b> camera models.
$p_1, p_2$	No	Number	Tangential distortion coefficients. Supported for <b>pinhole</b> camera models.
skew	No	Number	A parameter to measure any known skew in the image.
position	Yes	JSON object <b>Required Parameters:</b> x, y, and z. Enter numbers for these parameters.	The location or origin of the frame of reference of the camera mounted on the vehicle capturing images.

Parameter	Required	Accepted Values	Description
heading	Yes	JSON Object  <b>Required Parameters:</b>  $qx$ , $qy$ , $qz$ , and $qw$ . Enter numbers for these parameters.	The orientation of the frame of reference of the camera mounted on the vehicle capturing images, measured using <a href="#">quaternions</a> , ( $qx$ , $qy$ , $qz$ , $qw$ ).

### Sequence File and Point Cloud Frame Limits

You can include up to 100,000 point cloud frame sequences in your input manifest file. You can include up to 500 point cloud frames in each sequence file.

Keep in mind that 3D point cloud labeling job have longer pre-processing times than other Ground Truth task types. For more information, see [Job Pre-processing Time \(p. 259\)](#).

### Understand Coordinate Systems and Sensor Fusion

Point cloud data is always located in a coordinate system. This coordinate system may be local to the vehicle or the device sensing the surroundings, or it may be a world coordinate system. When you use Ground Truth 3D point cloud labeling jobs, all the annotations are generated using the coordinate system of your input data. For some labeling job task types and features, you must provide data in a world coordinate system.

In this topic, you'll learn the following:

- When you *are required* to provide input data in a world coordinate system or global frame of reference.
- What a world coordinate is and how you can convert point cloud data to a world coordinate system.
- How you can use your sensor and camera extrinsic matrices to provide pose data when using sensor fusion.

### Coordinate System Requirements for Labeling Jobs

If your point cloud data was collected in a local coordinate system, you can use an extrinsic matrix of the sensor used to collect the data to convert it to a world coordinate system or a global frame of reference. If you cannot obtain an extrinsic for your point cloud data and, as a result, cannot obtain point clouds in a world coordinate system, you can provide point cloud data in a local coordinate system for 3D point cloud object detection and semantic segmentation task types.

For object tracking, you must provide point cloud data in a world coordinate system. This is because when you are tracking objects across multiple frames, the ego vehicle itself is moving in the world and so all of the frames need a point of reference.

If you include camera data for sensor fusion, it is recommended that you provide camera poses in the same world coordinate system as the 3D sensor (such as a LiDAR sensor).

### Using Point Cloud Data in a World Coordinate System

This section explains what a world coordinate system (WCS), also referred to as a *global frame of reference*, is and explains how you can provide point cloud data in a world coordinate system.

#### What is a World Coordinate System?

A WCS or global frame of reference is a fixed universal coordinate system in which vehicle and sensor coordinate systems are placed. For example, if multiple point cloud frames are located in different

coordinate systems because they were collected from two sensors, a WCS can be used to translate all of the coordinates in these point cloud frames into a single coordinate system, where all frames have the same origin, (0,0,0). This transformation is done by translating the origin of each frame to the origin of the WCS using a translation vector, and rotating the three axes (typically x, y, and z) to the right orientation using a rotation matrix. This rigid body transformation is called a *homogeneous transformation*.

A world coordinate system is important in global path planning, localization, mapping, and driving scenario simulations. Ground Truth uses the right-handed Cartesian world coordinate system such as the one defined in [ISO 8855](#), where the x axis is forward toward the car's movement, y axis is left, and the z axis points up from the ground.

The global frame of reference depends on the data. Some datasets use the LiDAR position in the first frame as the origin. In this scenario, all the frames use the first frame as a reference and device heading and position will be near the origin in the first frame. For example, KITTI datasets have the first frame as a reference for world coordinates. Other datasets use a device position that is different from the origin.

Note that this is not the GPS/IMU coordinate system, which is typically rotated by 90 degrees along the z-axis. If your point cloud data is in a GPS/IMU coordinate system (such as OxTS in the open source AV KITTI dataset), then you need to transform the origin to a world coordinate system (typically the vehicle's reference coordinate system). You apply this transformation by multiplying your data with transformation metrics (the rotation matrix and translation vector). This will transform the data from its original coordinate system to a global reference coordinate system. Learn more about this transformation in the next section.

### Convert 3D Point Cloud Data to a WCS

Ground Truth assumes that your point cloud data has already been transformed into a reference coordinate system of your choice. For example, you can choose the reference coordinate system of the sensor (such as LiDAR) as your global reference coordinate system. You can also take point clouds from various sensors and transform them from the sensor's view to the vehicle's reference coordinate system view. You use the a sensor's extrinsic matrix, made up of a rotation matrix and translation vector, to convert your point cloud data to a WCS or global frame of reference.

Collectively, the translation vector and rotation matrix can be used to make up an *extrinsic matrix*, which can be used to convert data from a local coordinate system to a WCS. For example, your LiDAR extrinsic matrix may be composed as follows, where R is the rotation matrix and T is the translation vector:

```
LiDAR_extrinsic = [R T;0 0 0 1]
```

For example, the autonomous driving KITTI dataset includes a rotation matrix and translation vector for the LiDAR extrinsic transformation matrix for each frame. The [pykitti](#) python module can be used for loading the KITTI data, and in the dataset `dataset.oxts[i].T_w_imu` gives the LiDAR extrinsic transform for the  $i^{\text{th}}$  frame which can be multiplied with points in that frame to convert them to a world frame - `np.matmul(lidar_transform_matrix, points)`. Multiplying a point in LiDAR frame with a LiDAR extrinsic matrix transforms it into world coordinates. Multiplying a point in the world frame with the camera extrinsic matrix gives the point coordinates in the camera's frame of reference.

The following code example demonstrates how you can convert point cloud frames from the KITTI dataset into a WCS.

```
import pykitti
import numpy as np

basedir = '/Users/nameofuser/kitti-data'
date = '2011_09_26'
drive = '0079'
```

```
# The 'frames' argument is optional - default: None, which loads the whole dataset.
# Calibration, timestamps, and IMU data are read automatically.
# Camera and velodyne data are available via properties that create generators
# when accessed, or through getter methods that provide random access.
data = pykitti.raw(basedir, date, drive, frames=range(0, 50, 5))

# i is frame number
i = 0

# lidar extrinsic for the ith frame
lidar_extrinsic_matrix = data.oxts[i].T_w_imu

# velodyne raw point cloud in lidar scanners own coordinate system
points = data.get_velo(i)

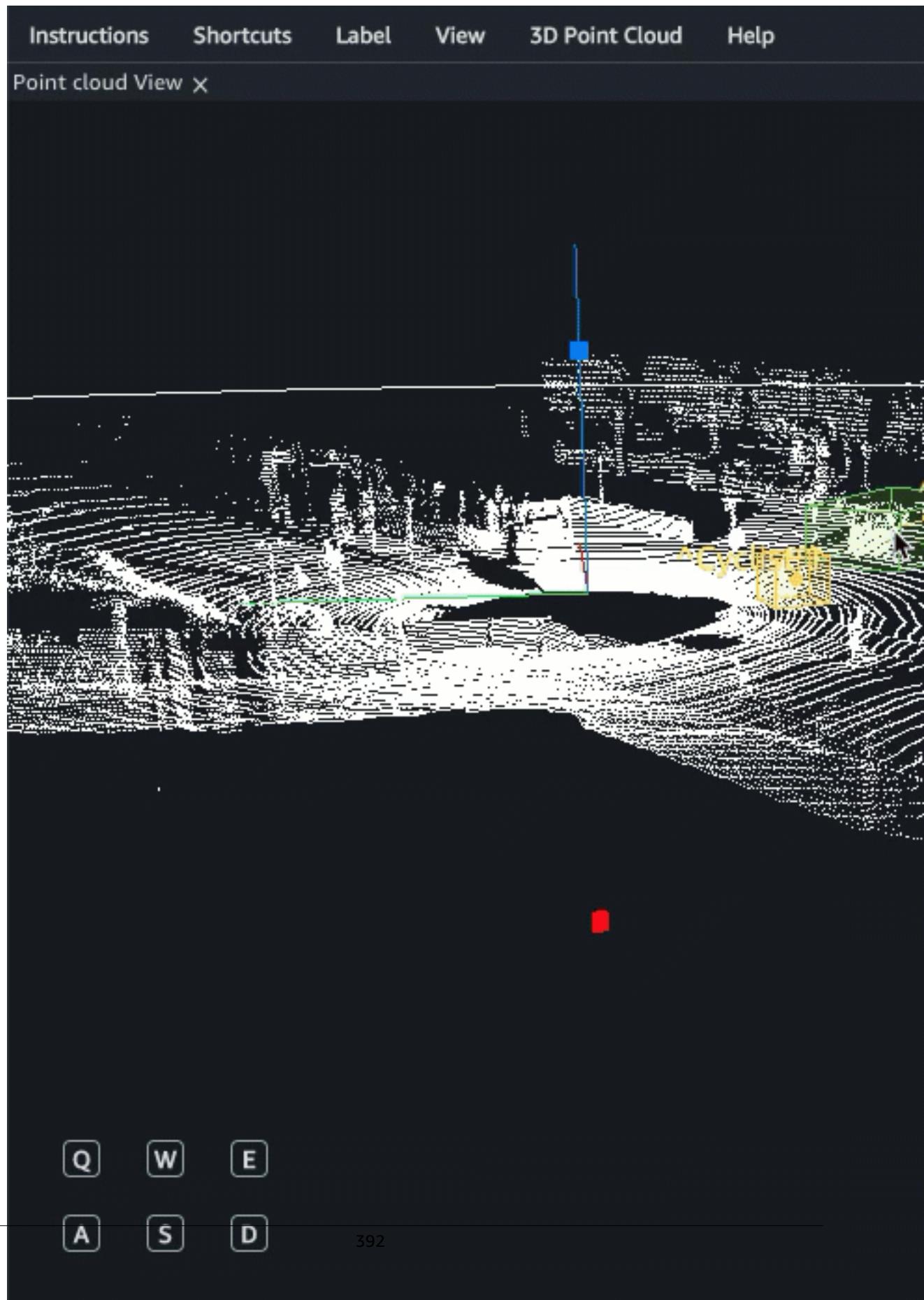
# transform points from lidar to global frame using lidar_extrinsic_matrix
def generate_transformed_pcd_from_point_cloud(points, lidar_extrinsic_matrix):
    tps = []
    for point in points:
        transformed_points = np.matmul(lidar_extrinsic_matrix, np.array([point[0],
    point[1], point[2], 1], dtype=np.float32).reshape(4,1)).tolist()
        if len(point) > 3 and point[3] is not None:
            tps.append([transformed_points[0][0], transformed_points[1][0],
    transformed_points[2][0], point[3]])

    return tps

# customer transforms points from lidar to global frame using lidar_extrinsic_matrix
transformed_pcl = generate_transformed_pcd_from_point_cloud(points, lidar_extrinsic_matrix)
```

## Sensor Fusion

Ground Truth supports sensor fusion of point cloud data with up to 8 video camera inputs. This feature allows human labellers to view the 3D point cloud frame side-by-side with the synchronized video frame. In addition to providing more visual context for labeling, sensor fusion allows workers to adjust annotations in the 3D scene and in 2D images and the adjustment are projected into the other view. The following video demonstrates a 3D point cloud labeling job with LiDAR and camera sensor fusion.



For best results, when using sensor fusion, your point cloud should be in a WCS. Ground Truth uses your sensor (such as LiDAR), camera, and ego vehicle pose information to compute extrinsic and intrinsic matrices for sensor fusion.

### Extrinsic Matrix

Ground Truth uses sensor (such as LiDAR) extrinsic and camera extrinsic and intrinsic matrices to project objects to and from the point cloud data's frame of reference to the camera's frame of reference.

For example, in order to project a label from the 3D point cloud to camera image plane, Ground Truth transforms 3D points from LiDAR's own coordinate system to the camera's coordinate system. This is typically done by first transforming 3D points from LiDAR's own coordinate system to a world coordinate system (or a global reference frame) using the LiDAR extrinsic matrix. Ground Truth then uses the camera inverse extrinsic (which converts points from a global frame of reference to the camera's frame of reference) to transform the 3D points from world coordinate system obtained in previous step into the camera image plane. The LiDAR extrinsic matrix can also be used to transform 3D data into a world coordinate system. If your 3D data is already transformed into world coordinate system then the first transformation doesn't have any impact on label translation, and label translation only depends on the camera inverse extrinsic. A view matrix is used to visualize projected labels. To learn more about these transformations and the view matrix, see [Ground Truth Sensor Fusion Transformations \(p. 397\)](#).

Ground Truth computes these extrinsic matrices by using LiDAR and camera *pose data* that you provide: heading (in quaternions: qx, qy, qz, and qw) and position (x, y, z). For the vehicle, typically the heading and position are described in vehicle's reference frame in a world coordinate system and are called a *ego vehicle pose*. For each camera extrinsic, you can add pose information for that camera. For more information, see [Pose \(p. 394\)](#).

### Intrinsic Matrix

Ground Truth use the camera extrinsic and intrinsic matrices to compute view metrics to transform labels to and from the 3D scene to camera images. Ground Truth computes the camera intrinsic matrix using camera focal length ( $f_x, f_y$ ) and optical center coordinates ( $c_x, c_y$ ) that you provide. For more information, see [Intrinsic and Distortion \(p. 397\)](#).

### Image Distortion

Image distortion can occur for a variety of reasons. For example, images may be distorted due to barrel or fish-eye effects. Ground Truth uses intrinsic parameters along with distortion co-efficient to undistort images you provide when creating 3D point cloud labeling jobs. If a camera image is already been undistorted, all distortion coefficients should be set to 0.

For more information about the transformations Ground Truth performs to undistort images, see [Camera Calibrations: Extrinsic, Intrinsic and Distortion \(p. 397\)](#).

### Ego Vehicle

To collect data for autonomous driving applications, the measurements used to generate point cloud data and are taken from sensors mounted on a vehicle, or the *ego vehicle*. To project label adjustments to and from the 3D scene and 2D images, Ground Truth needs your ego vehicle pose in a world coordinate system. The ego vehicle pose is comprised of position coordinates and orientation quaternion.

Ground Truth uses your ego vehicle pose to compute rotation and transformations matrices. Rotations in 3 dimensions can be represented by a sequence of 3 rotations around a sequence of axes. In theory, any three axes spanning the 3D Euclidean space are enough. In practice, the axes of rotation are chosen to be the basis vectors. The three rotations are expected to be in a global frame of reference (extrinsic). Ground Truth does not support body centered frame of reference (intrinsic) which is attached to, and moves with, the object under rotation. To track objects, Ground Truth needs to measure from a global reference where all vehicles are moving. When using Ground Truth 3D point cloud labeling jobs, z specifies the axis of rotation (extrinsic rotation) and yaw Euler angles are in radians (rotation angle).

## Pose

Ground Truth uses pose information for 3D visualizations and sensor fusion. Pose information you input through your manifest file is used to compute extrinsic matrices. If you already have an extrinsic matrix, you can use it to extract sensor and camera pose data.

For example in the autonomous driving KITTI dataset, the [pykitti](#) python module can be used for loading the KITTI data. In the dataset `dataset.oxts[i].T_w_imu` gives the LiDAR extrinsic transform for the  $i^{\text{th}}$  frame and it can be multiplied with the points to get them in a world frame - `matmul(lidar_transform_matrix, points)`. This transform can be converted into position (translation vector) and heading (in quaternion) of LiDAR for the input manifest file JSON format. Camera extrinsic transform for `cam0` in  $i^{\text{th}}$  frame can be calculated by `inv(matmul(dataset.calib.T_cam0_velo, inv(dataset.oxts[i].T_w_imu)))` and this can be converted into heading and position for `cam0`.

```
import numpy

rotation = [[ 9.96714314e-01, -8.09890350e-02,  1.16333982e-03],
            [ 8.09967396e-02,  9.96661051e-01, -1.03090934e-02],
            [-3.24531964e-04,  1.03694477e-02,  9.99946183e-01]]

origin= [1.71104606e+00,
         5.80000039e-01,
         9.43144935e-01]

from scipy.spatial.transform import Rotation as R

# position is the origin
position = origin
r = R.from_matrix(np.asarray(rotation))

# heading in WCS using scipy
heading = r.as_quat()
print(f"pose:{position}\nheading: {heading}")
```

## Position

In the input manifest file, `position` refers to the position of the sensor with respect to a world frame. If you are unable to put the device position in a world coordinate system, you can use LiDAR data with local coordinates. Similarly, for mounted video cameras you can specify the position and heading in a world coordinate system. For camera, if you do not have position information, please use (0, 0, 0).

The following are the fields in the position object:

1. `x` (float) – x coordinate of ego vehicle, sensor, or camera position in meters.
2. `y` (float) – y coordinate of ego vehicle, sensor, or camera position in meters.
3. `z` (float) – z coordinate of ego vehicle, sensor, or camera position in meters.

The following is an example of a `position` JSON object:

```
{
    "position": {
        "y": -152.77584902657554,
        "x": 311.21505956090624,
        "z": -10.854137529636024
    }
}
```

## Heading

In the input manifest file, `heading` is an object that represents the orientation of a device with respect to world frame. Heading values should be in quaternion. A [quaternion](#) is a representation of the orientation consistent with geodesic spherical properties. If you are unable to put the sensor heading in world coordinates, please use the identity quaternion ( $qx = 0$ ,  $qy = 0$ ,  $qz = 0$ ,  $qw = 1$ ). Similarly, for cameras, specify the heading in quaternions. If you are unable to obtain extrinsic camera calibration parameters, please also use the identity quaternion.

Fields in `heading` object are as follows:

1. `qx` (float) - x component of ego vehicle, sensor, or camera orientation.
2. `qy` (float) - y component of ego vehicle, sensor, or camera orientation.
3. `qz` (float) - z component of ego vehicle, sensor, or camera orientation.
4. `qw` (float) - w component of ego vehicle, sensor, or camera orientation.

The following is an example of a `heading` JSON object:

```
{  
    "heading": {  
        "qy": -0.7046155108831117,  
        "qx": 0.034278837280808494,  
        "qz": 0.7070617895701465,  
        "qw": -0.04904659893885366  
    }  
}
```

To learn more, see [Compute Orientation Quaternions and Position \(p. 395\)](#).

## Compute Orientation Quaternions and Position

Ground Truth requires that all orientation, or heading, data be given in quaternions. A [quaternions](#) is a representation of the orientation consistent with geodesic spherical properties that can be used to approximate of rotation. Compared to [Euler angles](#) they are simpler to compose and avoid the problem of [gimbal lock](#). Compared to rotation matrices they are more compact, more numerically stable, and more efficient.

You can compute quaternions from a rotation matrix or a transformation matrix.

If you have a rotation matrix (made up of the axis rotations) and translation vector (or origin) in world coordinate system instead of a single 4x4 rigid transformation matrix, then you can directly use the rotation matrix and translation vector to compute quaternions. Libraries like [scipy](#) and [pyquatnion](#) can help. The following code-block shows an example using these libraries to compute quaternion from a rotation matrix.

```
import numpy  
  
rotation = [[ 9.96714314e-01, -8.09890350e-02,  1.16333982e-03],  
            [ 8.09967396e-02,  9.96661051e-01, -1.03090934e-02],  
            [-3.24531964e-04,  1.03694477e-02,  9.99946183e-01]]  
  
origin = [1.71104606e+00,  
          5.80000039e-01,  
          9.43144935e-01]  
  
from scipy.spatial.transform import Rotation as R  
# position is the origin  
position = origin
```

```
r = R.from_matrix(np.asarray(rotation))
# heading in WCS using scipy
heading = r.as_quat()
print(f"position:{position}\nheading: {heading}")
```

A UI tool like [3D Rotation Converter](#) can also be useful.

If you have a 4x4 extrinsic transformation matrix, note that the transformation matrix is in the form [R T; 0 0 0 1] where R is the rotation matrix and T is the origin translation vector. That means you can extract rotation matrix and translation vector from the transformation matrix as follows.

```
import numpy as np

transformation
= [[ 9.96714314e-01, -8.09890350e-02,  1.16333982e-03,  1.71104606e+00],
   [ 8.09967396e-02,  9.96661051e-01, -1.03090934e-02,  5.80000039e-01],
   [-3.24531964e-04,  1.03694477e-02,  9.99946183e-01,  9.43144935e-01],
   [ 0, 0, 0, 1]]

transformation = np.array(transformation )
rotation = transformation[0:3][0:3]
translation= transformation[0:3][3]

from scipy.spatial.transform import Rotation as R
# position is the origin translation
position = translation
r = R.from_matrix(np.asarray(rotation))
# heading in WCS using scipy
heading = r.as_quat()
print(f"position:{position}\nheading: {heading}")
```

With your own setup, you can compute an extrinsic transformation matrix using the GPS/IMU position and orientation (latitude, longitude, altitude and roll, pitch, yaw) with respect to the LiDAR sensor on the ego vehicle. For example, you can compute pose from KITTI raw data using `pose = convertOxtsToPose(oxts)` to transform the oxts data into a local euclidean poses, specified by 4x4 rigid transformation matrices. You can then transform this pose transformation matrix to a global reference frame using the reference frames transformation matrix in the world coordinate system.

```
struct Quaternion
{
    double w, x, y, z;
};

Quaternion ToQuaternion(double yaw, double pitch, double roll) // yaw (Z), pitch (Y), roll (X)
{
    // Abbreviations for the various angular functions
    double cy = cos(yaw * 0.5);
    double sy = sin(yaw * 0.5);
    double cp = cos(pitch * 0.5);
    double sp = sin(pitch * 0.5);
    double cr = cos(roll * 0.5);
    double sr = sin(roll * 0.5);

    Quaternion q;
    q.w = cr * cp * cy + sr * sp * sy;
    q.x = sr * cp * cy - cr * sp * sy;
    q.y = cr * sp * cy + sr * cp * sy;
    q.z = cr * cp * sy - sr * sp * cy;

    return q;
}
```

## Ground Truth Sensor Fusion Transformations

The following sections go into greater detail about the Ground Truth sensor fusion transformations that are performed using the pose data you provide.

### LiDAR Extrinsic

In order to project to and from a 3D LiDAR scene to a 2D camera image, Ground Truth computes the rigid transformation projection metrics using the ego vehicle pose and heading. Ground Truth computes rotation and translation of a world coordinates into the 3D plane by doing a simple sequence of rotations and translation.

Ground Truth computes rotation metrics using the heading quaternions as follows:

$$M = \begin{pmatrix} 1 - 2y^2 - 2z^2 & 2xy + 2zw & 2xz - 2yw \\ 2xy - 2zw & 1 - 2x^2 - 2z^2 & 2yz + 2xw \\ 2xz + 2yw & 2yz - 2xw & 1 - 2x^2 - 2y^2 \end{pmatrix}$$

Here,  $[x, y, z, w]$  corresponds to parameters in the heading JSON object,  $[qx, qy, qz, qw]$ . Ground Truth computes the translation column vector as  $T = [poseX, poseY, poseZ]$ . Then the extrinsic metrics is simply as follows:

```
LiDAR_extrinsic = [R T;0 0 0 1]
```

### Camera Calibrations: Extrinsic, Intrinsic and Distortion

*Geometric camera calibration*, also referred to as *camera resectioning*, estimates the parameters of a lens and image sensor of an image or video camera. You can use these parameters to correct for lens distortion, measure the size of an object in world units, or determine the location of the camera in the scene. Camera parameters include intrinsics and distortion coefficients.

### Camera Extrinsic

If the camera pose is given, then Ground Truth computes the camera extrinsic based on a rigid transformation from the 3D plane into the camera plane. The calculation is the same as the one used for the [LiDAR Extrinsic \(p. 397\)](#), except that Ground Truth uses camera pose (position and heading) and computes the inverse extrinsic.

```
camera_inverse_extrinsic = inv([Rc Tc;0 0 0 1]) #where Rc and Tc are camera pose components
```

### Intrinsic and Distortion

Some cameras, such as pinhole or fisheye cameras, may introduce significant distortion in photos. This distortion can be corrected using distortion coefficients and the camera focal length. To learn more, see [Camera Calibration With OpenCV](#) in the OpenCV documentation.

There are two types of distortion Ground Truth can correct for: radial distortion and tangential distortion.

*Radial distortion* occurs when light rays bend more near the edges of a lens than they do at its optical center. The smaller the lens, the greater the distortion. The presence of the radial distortion manifests in form of the *barrel* or *fish-eye* effect and Ground Truth uses Formula 1 to undistort it.

#### Formula 1:

$$\begin{aligned} x_{corrected} &= x(1 + k_1r^2 + k_2r^4 + k_3r^6) \\ y_{corrected} &= y(1 + k_1r^2 + k_2r^4 + k_3r^6) \end{aligned}$$

*Tangential distortion* occurs because the lenses used to take the images are not perfectly parallel to the imaging plane. This can be corrected with Formula 2.

**Formula 2:**

$$x_{corrected} = x + [2p_1xy + p_2(r^2 + 2x^2)] \\ y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2xy]$$

In the input manifest file, you can provide distortion coefficients and Ground Truth will undistort your images. All distortion coefficients are floats.

- $k_1, k_2, k_3, k_4$  – Radial distortion coefficients. Supported for both fisheye and pinhole camera models.
- $p_1, p_2$  – Tangential distortion coefficients. Supported for pinhole camera models.

If images are already undistorted, all distortion coefficients should be 0 in your input manifest.

In order to correctly reconstruct the corrected image, Ground Truth does a unit conversion of the images based on focal lengths. If a common focal length is used with a given aspect ratio for both axes, such as 1, in the upper formula we will have a single focal length. The matrix containing these four parameters is referred to as the *in camera intrinsic calibration matrix*.

$$\begin{Bmatrix} x \\ y \\ w \end{Bmatrix} = \begin{Bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{Bmatrix} \begin{Bmatrix} X \\ Y \\ Z \end{Bmatrix}$$

While the distortion coefficients are the same regardless of the camera resolutions used, these should be scaled with the current resolution from the calibrated resolution.

The following are float values.

- $f_x$  - focal length in x direction.
- $f_y$  - focal length in y direction.
- $c_x$  - x coordinate of principal point.
- $c_y$  - y coordinate of principal point.

Ground Truth use the camera extrinsic and camera intrinsic to compute view metrics as shown in the following code block to transform labels between the 3D scene and 2D images.

```
def generate_view_matrix(intrinsic_matrix, extrinsic_matrix):
    intrinsic_matrix = np.c_[intrinsic_matrix, np.zeros(3)]
    view_matrix = np.matmul(intrinsic_matrix, extrinsic_matrix)
    view_matrix = np.insert(view_matrix, 2, np.array((0, 0, 0, 1)), 0)
    return view_matrix
```

## Video Frame Input Data

When you create a video frame object detection or object tracking labeling job, you can choose video files (MP4 files) or video frames for input data. All worker tasks are created using video frames, so if you choose video files, use the Ground Truth frame extraction tool to extract video frames (images) from your video files.

For both of these options, you can use the **Automated data setup** option in the Ground Truth section of the Amazon SageMaker console to set up a connection between Ground Truth and your input data in Amazon S3 so that Ground Truth knows where to look for your input data when creating your labeling tasks. This creates and stores an input manifest file in your Amazon S3 input dataset location. To learn more, see [Automated Video Frame Input Data Setup \(p. 401\)](#).

Alternatively, you can manually create sequence files for each sequence of video frames that you want labeled and provide the Amazon S3 location of an input manifest file that references each of these sequences files using the `source-ref` key. To learn more, see [Create a Video Frame Input Manifest File \(p. 402\)](#).

### Topics

- [Choose Video Files or Video Frames for Input Data \(p. 399\)](#)
- [Input Data Setup \(p. 400\)](#)

## Choose Video Files or Video Frames for Input Data

When you create a video frame object detection or object tracking labeling job, you can provide a sequence of video frames (images) or you can use the Amazon SageMaker console to have Ground Truth automatically extract video frames from your video files. Use the following sections to learn more about these options.

### Provide Video Frames

Video frames are sequences of images extracted from a video file. You can create a Ground Truth labeling job to have workers label multiple sequences of video frames. Each sequence is made up of images extracted from a single video.

To create a labeling job using video frame sequences, you must store each sequence using a unique **key name prefix** in Amazon S3. In the Amazon S3 console, key name prefixes are folders. So in the Amazon S3 console, each sequence of video frames must be located in its own folder in Amazon S3.

For example, if you have two sequences of video frames, you might use the key name prefixes `sequence1/` and `sequence2/` to identify your sequences. In this example, your sequences may be located in `s3://video-frames/sequence1/` and `s3://video-frames/sequence2/`.

If you are using the Ground Truth console to create an input manifest file, all of the sequence key name prefixes should be in the same location in Amazon S3. For example, in the Amazon S3 console, each sequence could be in a folder in `s3://video-frames/`. In this example, your first sequence of video frames (images) may be located in `s3://video-frames/sequence1/` and your second sequence may be located in `s3://video-frames/sequence2/`.

### Important

Even if you only have a single sequence of video frames that you want workers to label, that sequence must have a key name prefix in Amazon S3. If you are using the Amazon S3 console, this means that your sequence is located in a folder. It cannot be located in the root of your S3 bucket.

When creating worker tasks using sequences of video frames, Ground Truth uses one sequence per task. In each task, Ground Truth orders your video frames using [UTF-8](#) binary order.

For example, video frames might be in the following order in Amazon S3:

```
[0001.jpg, 0002.jpg, 0003.jpg, ..., 0011.jpg]
```

They are arranged in the same order in the worker's task: `0001.jpg`, `0002.jpg`, `0003.jpg`, ..., `0011.jpg`.

Frames might also be ordered using a naming convention like the following:

```
[frame1.jpg, frame2.jpg, ..., frame11.jpg]
```

In this case, `frame10.jpg` and `frame11.jpg` come before `frame2.jpg` in the worker task. Your worker sees your video frames in the following order: `frame1.jpg, frame10.jpg, frame11.jpg, frame2.jpg, ..., frame9.jpg`.

### Provide Video Files

You can use the Ground Truth frame splitting feature when creating a new labeling job in the console to extract video frames from video files (MP4 files). A series of video frames extracted from a single video file is referred to as a *sequence of video frames*.

You can either have Ground Truth automatically extract all frames, up to 2,000, from the video, or you can specify a frequency for frame extraction. For example, you can have Ground Truth extract every 10<sup>th</sup> frame from your videos.

You can provide up to 50 videos when you use automated data setup to extract frames, however your input manifest file cannot reference more than 10 video frame sequence files when you create a video frame object tracking and video frame object detection labeling job. If you use the automated data setup console tool to extract video frames from more than 10 video files, you will need to modify the manifest file the tool generates or create a new one to include 10 video frame sequence files or less. To learn more about these quotas, see [3D Point Cloud and Video Frame Labeling Job Quotas \(p. 373\)](#).

To use the video frame extraction tool, see [Automated Video Frame Input Data Setup \(p. 401\)](#).

When all of your video frames have been successfully extracted from your videos, you will see the following in your S3 input dataset location:

- A key name prefix (a folder in the Amazon S3 console) named after each video. Each of these prefixes leads to:
  - A sequence of video frames extracted from the video used to name that prefix.
  - A sequence file used to identify all of the images that make up that sequence.
- An input manifest file with a .manifest extension. This identifies all of the sequence files that will be used to create your labeling job.

All of the frames extracted from a single video file are used for a labeling task. If you extract video frames from multiple video files, multiple tasks are created for your labeling job, one for each sequence of video frames.

Ground Truth stores each sequence of video frames that it extracts in your Amazon S3 location for input datasets using a unique [key name prefix](#). In the Amazon S3 console, key name prefixes are folders.

### Input Data Setup

When you create a video frame labeling job, you need to let Ground Truth know where to look for your input data. You can do this in one of two ways:

- You can store your input data in Amazon S3 and have Ground Truth automatically detect the input dataset used for your labeling job. See [Automated Video Frame Input Data Setup \(p. 401\)](#) to learn more about this option.
- You can create an input manifest file and sequence files and upload them to Amazon S3. See [Manual Input Data Setup \(p. 402\)](#) to learn more about this option.

### Topics

- [Automated Video Frame Input Data Setup \(p. 401\)](#)
- [Manual Input Data Setup \(p. 402\)](#)

## Automated Video Frame Input Data Setup

You can use the Ground Truth automated data setup to automatically detect video files in your Amazon S3 bucket and extract video frames from those files. To learn how, see [Provide Video Files \(p. 400\)](#).

If you already have video frames in Amazon S3, you can use the automated data setup to use these video frames in your labeling job. For this option, all video frames from a single video must be stored using a unique prefix. To learn about the requirements to use this option, see [Provide Video Frames \(p. 399\)](#).

Select one of the following sections to learn how to set up your automatic input dataset connection with Ground Truth.

### Provide Video Files and Extract Frames

Use the following procedure to connect your video files with Ground Truth and automatically extract video frames from those files for video frame object detection and object tracking labeling jobs.

#### Note

If you use the automated data setup console tool to extract video frames from more than 10 video files, you will need to modify the manifest file the tool generates or create a new one to include 10 video frame sequence files or less. To learn more, see [Provide Video Files \(p. 400\)](#).

Make sure your video files are stored in an Amazon S3 bucket in the same Amazon Region that you perform the automated data setup in.

### Automatically connect your video files in Amazon S3 with Ground Truth and extract video frames:

1. Navigate to the **Create labeling job** page in the Amazon SageMaker console: <https://console.amazonaws.cn/sagemaker/groundtruth>.

Your input and output S3 buckets must be located in the same Amazon Region that you create your labeling job in. This link puts you in the North Virginia (us-east-1) Amazon Region. If your input data is in an Amazon S3 bucket in another Region, switch to that Region. To change your Amazon Region, on the **navigation bar**, choose the name of the currently displayed Region.

2. Select **Create labeling job**.
3. Enter a **Job name**.
4. In the section **Input data setup**, select **Automated data setup**.
5. Enter an Amazon S3 URI for **S3 location for input datasets**. An S3 URI looks like the following: `s3://path-to-files/`. This URI should point to the Amazon S3 location where your video files are stored.
6. Specify your **S3 location for output datasets**. This is where your output data is stored. You can choose to store your output data in the **Same location as input dataset** or **Specify a new location** and entering the S3 URI of the location that you want to store your output data.
7. Choose **Video Files** for your **Data type** using the dropdown list.
8. Choose **Yes, extract frames for object tracking and detection tasks**.
9. Choose a method of **Frame extraction**.
  - When you choose **Use all frames extracted from the video to create a labeling task**, Ground Truth extracts all frames from each video in your **S3 location for input datasets**, up to 2,000 frames. If a video in your input dataset contains more than 2,000 frames, the first 2,000 are extracted and used for that labeling task.
  - When you choose **Use every  $x$  frame from a video to create a labeling task**, Ground Truth extracts every  $x^{\text{th}}$  frame from each video in your **S3 location for input datasets**.

For example, if your video is 2 seconds long, and has a **frame rate** of 30 frames per second, there are 60 frames in your video. If you specify 10 here, Ground Truth extracts every 10<sup>th</sup> frame from your video. This means the 1<sup>st</sup>, 10<sup>th</sup>, 20<sup>th</sup>, 30<sup>th</sup>, 40<sup>th</sup>, 50<sup>th</sup>, and 60<sup>th</sup> frames are extracted.

10. Choose or create an IAM execution role. Make sure that this role has permission to access your Amazon S3 locations for input and output data specified in steps 5 and 6.
11. Select **Complete data setup**.

### Provide Video Frames

Use the following procedure to connect your sequences of video frames with Ground Truth for video frame object detection and object tracking labeling jobs.

Make sure your video frames are stored in an Amazon S3 bucket in the same Amazon Region that you perform the automated data setup in. Each sequence of video frames should have a unique prefix. For example, if you have two sequences stored in `s3://video-frames/sequences/`, each should have a unique prefix like `sequence1` and `sequence2` and should both be located directly under the `/sequences/` prefix. In the example above, the locations of these two sequences is: `s3://video-frames/sequences/sequence1/` and `s3://video-frames/sequences/sequence2/`.

#### Automatically connect your video frame in Amazon S3 with Ground Truth:

1. Navigate to the **Create labeling job** page in the Amazon SageMaker console: <https://console.amazonaws.cn/sagemaker/groundtruth>.

Your input and output S3 buckets must be located in the same Amazon Region that you create your labeling job in. This link puts you in the North Virginia (us-east-1) Amazon Region. If your input data is in an Amazon S3 bucket in another Region, switch to that Region. To change your Amazon Region, on the [navigation bar](#), choose the name of the currently displayed Region.

2. Select **Create labeling job**.
3. Enter a **Job name**.
4. In the section **Input data setup**, select **Automated data setup**.
5. Enter an Amazon S3 URI for **S3 location for input datasets**.

This should be the Amazon S3 location where your sequences are stored. For example, if you have two sequences stored in `s3://video-frames/sequences/sequence1/`, `s3://video-frames/sequences/sequence2/`, enter `s3://video-frames/sequences/` here.

6. Specify your **S3 location for output datasets**. This is where your output data is stored. You can choose to store your output data in the **Same location as input dataset** or **Specify a new location** and entering the S3 URI of the location that you want to store your output data.
7. Choose **Video frames** for your **Data type** using the dropdown list.
8. Choose or create an IAM execution role. Make sure that this role has permission to access your Amazon S3 locations for input and output data specified in steps 5 and 6.
9. Select **Complete data setup**.

These procedures will create an input manifest in the Amazon S3 location for input datasets that you specified in step 5. If you are creating a labeling job using the SageMaker API or, Amazon CLI, or an Amazon SDK, use the Amazon S3 URI for this input manifest file as input to the parameter `ManifestS3Uri`.

### Manual Input Data Setup

Choose the manual data setup option if you have created sequence files for each of your video frame sequences, and a manifest file listing references to those sequences files.

#### Create a Video Frame Input Manifest File

Ground Truth uses the input manifest file to identify the location of your input dataset when creating labeling tasks. For video frame object detection and object tracking labeling jobs, each line in the input

manifest file identifies the location of a video frame sequence file. Each sequence file identifies the images included in a single sequence of video frames.

Use this page to learn how to create a video frame sequence file and an input manifest file for video frame object tracking and object detection labeling jobs.

If you want Ground Truth to automatically generate your sequence files and input manifest file, see [Automated Video Frame Input Data Setup \(p. 401\)](#).

### Create a Video Frame Sequence Input Manifest

In the video frame sequence input manifest file, each line in the manifest is a JSON object, with a "source-ref" key that references a sequence file. Each sequence file identifies the location of a sequence of video frames. This is the manifest file formatting required for all video frame labeling jobs.

The following example demonstrates the syntax used for an input manifest file:

```
{"source-ref": "s3:///example-folder/seq1.json"}  
{"source-ref": "s3:///example-folder/seq2.json"}
```

### Create a Video Frame Sequence File

The data for each sequence of video frames needs to be stored in a JSON data object. The following is an example of the format you use for a sequence file. Information about each frame is included as a JSON object and is listed in the `frames` list. The following JSON has been expanded for readability.

```
{
  "seq-no": 1,
  "prefix": "s3://mybucket/prefix/video1/",
  "number-of-frames": 3,
  "frames": [
    {"frame-no": 1, "unix-timestamp": 1566861644, "frame": "frame0001.jpg" },
    {"frame-no": 2, "unix-timestamp": 1566861644, "frame": "frame0002.jpg" },
    {"frame-no": 3, "unix-timestamp": 1566861644, "frame": "frame0003.jpg" }
  ]
}
```

The following table provides details about the parameters shown in the this code example.

Parameter	Required	Accepted Values	Description
seq-no	Yes	Integer	The ordered number of the sequence.
prefix	Yes	String <b>Accepted Values:</b> <code>s3://&lt;bucket-name&gt;/&lt;prefix&gt;/</code>	The Amazon S3 location where the sequence files are located.  The prefix must end with a forward slash: /.
number-of-frames	Yes	Integer	The total number of frames included in the sequence file. This number must match the total number of frames listed in the <code>frames</code>

Parameter	Required	Accepted Values	Description
			parameter in the next row.
<b>frames</b>	Yes	List of JSON objects  <b>Required:</b> <code>frame-no</code> , <code>frame</code>  <b>Optional:</b> <code>unix-timestamp</code>	A list of frame data. The length of the list must equal <code>number-of-frames</code> . In the worker UI, frames in a sequence are ordered in <a href="#">UTF-8</a> binary order. To learn more about this ordering, see <a href="#">Provide Video Frames (p. 399)</a> .
<b>frame-no</b>	Yes	Integer	The frame order number. This will determine the order of a frame in the sequence.
<b>unix-timestamp</b>	No	Integer	The unix timestamp of a frame. The number of seconds since January 1st, 1970 until the UTC time when the frame was captured.
<b>frame</b>	Yes	String	The name of a video frame image file.

## Output Data

The output from a labeling job is placed in the Amazon S3 location that you specified in the console or in the call to the [CreateLabelingJob](#) operation. Output data appears in this location when the workers have submitted one or more tasks, or when tasks expire. Note that it may take a few minutes for output data to appear in Amazon S3 after the worker submits the task or the task expires.

Each line in the output data file is identical to the manifest file with the addition of an attribute and value for the label assigned to the input object. The attribute name for the value is defined in the console or in the call to the [CreateLabelingJob](#) operation. You can't use `-metadata` in the label attribute name. If you are running an image semantic segmentation, 3D point cloud semantic segmentation, or 3D point cloud object tracking job, the label attribute must end with `-ref`. For any other type of job, the attribute name can't end with `-ref`.

The output of the labeling job is the value of the key-value pair with the label. The label and the value overwrites any existing JSON data in the input file with the new value.

For example, the following is the output from an image classification labeling job where the input data files were stored in an Amazon S3 [AWSDOC-EXAMPLE-BUCKET](#) and the label attribute name was defined as `sport`. In this example the JSON object is formatted for readability, in the actual output file the JSON object is on a single line. For more information about the data format, see [JSON Lines](#).

```
{
  "source-ref": "s3://AWSDOC-EXAMPLE-BUCKET/image_example.png",
  "sport": 0,
  "sport-metadata":
```

```
{  
    "class-name": "football",  
    "confidence": 0.00,  
    "type": "groundtruth/image-classification",  
    "job-name": "identify-sport",  
    "human-annotated": "yes",  
    "creation-date": "2018-10-18T22:18:13.527256"  
}  
}
```

The value of the label can be any valid JSON. In this case the label's value is the index of the class in the classification list. Other job types, such as bounding box, have more complex values.

Any key-value pair in the input manifest file other than the label attribute is unchanged in the output file. You can use this to pass data to your application.

The output from a labeling job can be used as the input to another labeling job. You can use this when you are chaining together labeling jobs. For example, you can send one labeling job to determine the sport that is being played. Then you send another using the same data to determine if the sport is being played indoors or outdoors. By using the output data from the first job as the manifest for the second job, you can consolidate the results of the two jobs into one output file for easier processing by your applications.

The output data file is written to the output location periodically while the job is in progress. These intermediate files contain one line for each line in the manifest file. If an object is labeled, the label is included. If the object hasn't been labeled, it is written to the intermediate output file identically to the manifest file.

## Output Directories

Ground Truth creates several directories in your Amazon S3 output path. These directories contain the results of your labeling job and other artifacts of the job. The top-level directory for a labeling job is given the same name as your labeling job; the output directories are placed beneath it. For example, if you named your labeling job **find-people**, your output would be in the following directories:

```
s3://AWSDOC-EXAMPLE-BUCKET/find-people/activelearning  
s3://AWSDOC-EXAMPLE-BUCKET/find-people/annotations  
s3://AWSDOC-EXAMPLE-BUCKET/find-people/inference  
s3://AWSDOC-EXAMPLE-BUCKET/find-people/manifests  
s3://AWSDOC-EXAMPLE-BUCKET/find-people/training
```

Each directory contains the following output:

### Active Learning Directory

The activelearning directory is only present when you are using automated data labeling. It contains the input and output validation set for automated data labeling, and the input and output folder for automatically labeled data.

### Annotations Directory

The annotations directory contains all of the annotations made by the workforce. These are the responses from individual workers that have not been consolidated into a single label for the data object.

There are three subdirectories in the annotations directory.

- The first, **worker-response**, contains the responses from individual workers. This contains a subdirectory for each iteration, which in turn contains a subdirectory for each data object in that iteration. The annotation for each data object is stored in a timestamped JSON file that contains the

annotation made by a single worker, and if you use a private workforce, metadata about that worker. To learn more about this metadata, see [Worker Metadata \(p. 406\)](#).

- The second, **consolidated-annotation**, contains information required to consolidate the annotations in the current batch into labels for your data objects.
- The third, **intermediate**, contains the output manifest for the current batch with any completed labels. This file is updated as the label for each data object is completed.

### Inference Directory

The **inference** directory is only present when you are using automated data labeling. This directory contains the input and output files for the SageMaker batch transform used while labeling data objects.

### Manifest Directory

The **manifest** directory contains the output manifest from your labeling job. There is one subdirectory in the manifest directory, **output**. The **output** directory contains the output manifest file for your labeling job. The file is named **output.manifest**.

### Training Directory

The **training** directory is only present when you are using automated data labeling. This directory contains the input and output files used to train the automated data labeling model.

### Confidence Score

When you have more than one worker annotate a single task, your label results from annotation consolidation. Ground Truth calculates a confidence score for each label. A **confidence score** is a number between 0 and 1 that indicates how confident Ground Truth is in the label. You can use the confidence score to compare labeled data objects to each other, and to identify the least or most confident labels.

You should not interpret the value of a confidence score as an absolute value, or compare confidence scores across labeling jobs. For example, if all of the confidence scores are between 0.98 and 0.998, you should only compare the data objects with each other and not rely on the high confidence scores.

You should not compare the confidence scores of human-labeled data objects and auto-labeled data objects. The confidence scores for humans are calculated using the annotation consolidation function for the task, while the confidence scores for automated labeling are calculated using a model that incorporates object features. The two models generally have different scales and average confidence.

For a bounding box labeling job, Ground Truth calculates a confidence score per box. You can compare confidence scores within one image or across images for the same labeling type (human or auto). You can't compare confidence scores across labeling jobs.

If a single worker annotates a task (`NumberOfHumanWorkersPerDataObject` is set to 1 or in the console, you enter 1 for **Number of workers per dataset object**), the confidence score is set to 0.00.

### Worker Metadata

Ground Truth provides information that you can use to track individual workers in task output data. The following data is located in the directories under the **worker-response** located in the [Annotations Directory \(p. 405\)](#):

- The `acceptanceTime` is the time that the worker accepted the task. The format of this date and time stamp is `YYYY-MM-DDTHH:MM:SS.mmmZ` for the year (YYYY), month (MM), day (DD), hour (HH), minute (MM), second (SS) and millisecond (mmm). The date and time are separated by a T.
- The `submissionTime` is the time that the worker submitted their annotations using the **Submit** button. The format of this date and time stamp is `YYYY-MM-DDTHH:MM:SS.mmmZ` for the year (YYYY), month (MM), day (DD), hour (HH), minute (MM), second (SS) and millisecond (mmm). The date and time are separated by a T.

- `timeSpentInSeconds` reports the total time, in seconds, that a worker actively worked on that task. This metric does not include time when a worker paused or took a break.
- The `workerId` is unique to each worker.
- If you use a [private workforce](#), in `workerMetadata`, you see the following.
  - The `identityProviderType` is the service used to manage the private workforce.
  - The `issuer` is the Cognito user pool or OIDC Identity Provider (IdP) issuer associated with the work team assigned to this human review task.
  - A unique `sub` identifier refers to the worker. If you create a workforce using Amazon Cognito, you can retrieve details about this worker (such as the name or user name) using this ID using Amazon Cognito. To learn how, see [Managing and Searching for User Accounts in Amazon Cognito Developer Guide](#).

The following is an example of the output you may see if you use Amazon Cognito to create a private workforce. This is identified in the `identityProviderType`.

```
"submissionTime": "2020-12-28T18:59:58.321Z",
"acceptanceTime": "2020-12-28T18:59:15.191Z",
"timeSpentInSeconds": 40.543,
"workerId": "a12b3cdefg4h5i67",
"workerMetadata": {
    "identityData": {
        "identityProviderType": "Cognito",
        "issuer": "https://cognito-idp.us-west-2.amazonaws.com/us-west-2_123456789",
        "sub": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee"
    }
}
```

The following is an example of the `workerMetadata` you may see if you use your own OIDC IdP to create a private workforce:

```
"workerMetadata": {
    "identityData": {
        "identityProviderType": "Oidc",
        "issuer": "https://example-oidc-ipd.com/adfs",
        "sub": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee"
    }
}
```

To learn more about using private workforces, see [Use a Private Workforce \(p. 461\)](#).

## Output Metadata

The output from each job contains metadata about the label assigned to data objects. These elements are the same for all jobs with minor variations. The following example shows the metadata elements:

```
"confidence": 0.00,
"type": "groundtruth/image-classification",
"job-name": "identify-animal-species",
"human-annotated": "yes",
"creation-date": "2020-10-18T22:18:13.527256"
```

The elements have the following meaning:

- `confidence` – The confidence that Ground Truth has that the label is correct. For more information, see [Confidence Score \(p. 406\)](#).
- `type` – The type of classification job. For job types, see [Built-in Task Types \(p. 333\)](#).

- **job-name** – The name assigned to the job when it was created.
- **human-annotated** – Whether the data object was labeled by a human or by automated data labeling. For more information, see [Automate Data Labeling \(p. 430\)](#).
- **creation-date** – The date and time that the label was created.

## Classification Job Output

The following are sample outputs (output manifest files) from an image classification job and a text classification job. They include the label that Ground Truth assigned to the data object, the value for the label, and metadata that describes the label.

In addition to the standard metadata elements, the metadata for a classification job includes the text value of the label's class. For more information, see [Image Classification Algorithm \(p. 1399\)](#).

The red, italicized text in the examples below depends on labeling job specifications and output data.

```
{
  "source-ref": "s3://AWSDOC-EXAMPLE-BUCKET/example_image.jpg",
  "species": "0",
  "species-metadata":
  {
    "class-name": "dog",
    "confidence": "0.00",
    "type": "groundtruth/image-classification",
    "job-name": "identify-animal-species",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256"
  }
}
```

```
{
  "source": "The food was delicious",
  "mood": "1",
  "mood-metadata":
  {
    "class-name": "positive",
    "confidence": "0.8",
    "type": "groundtruth/text-classification",
    "job-name": "label-sentiment",
    "human-annotated": "yes",
    "creation-date": "2020-10-18T22:18:13.527256"
  }
}
```

## Multi-label Classification Job Output

The following are example output manifest files from a multi-label image classification job and a multi-label text classification job. They include the labels that Ground Truth assigned to the data object (for example, the image or piece of text) and metadata that describes the labels the worker saw when completing the labeling task.

The label attribute name parameter (for example, `image-label-attribute-name`) contains an array of all of the labels selected by at least one of the workers who completed this task. This array contains integer keys (for example, `[1, 0, 8]`) that correspond to the labels found in `class-map`. In the multi-label image classification example, `bicycle`, `person`, and `clothing` were selected by at least one of the workers who completed the labeling task for the image, `exampleimage.jpg`.

The `confidence-map` shows the confidence score that Ground Truth assigned to each label selected by a worker. To learn more about Ground Truth confidence scores, see [Confidence Score \(p. 406\)](#).

The red, italicized text in the examples below depends on labeling job specifications and output data.

The following is an example of a multi-label image classification output manifest file.

```
{
  "source-ref": "s3://AWSDOC-EXAMPLE-BUCKET/example_image.jpg",
  "image-label-attribute-name": [1, 0, 8],
  "image-label-attribute-name-metadata":
  {
    "job-name": "labeling-job/image-label-attribute-name",
    "class-map":
    {
      "1": "bicycle", "0": "person", "8": "clothing"
    },
    "human-annotated": "yes",
    "creation-date": "2020-02-27T21:36:25.000Z01",
    "confidence-map":
    {
      "1": 0.95, "0": 0.77, "8": 0.2
    },
    "type": "groundtruth/image-classification-multilabel"
  }
}
```

The following is an example of a multi-label text classification output manifest file. In this example, `approving`, `sad` and `critical` were selected by at least one of the workers who completed the labeling task for the object `exampletext.txt` found in `AWSDOC-EXAMPLE-BUCKET`.

```
{
  "source-ref": "AWSDOC-EXAMPLE-BUCKET/text_file.txt",
  "text-label-attribute-name": [1, 0, 4],
  "text-label-attribute-name-metadata":
  {
    "job-name": "labeling-job/text-label-attribute-name",
    "class-map":
    {
      "1": "approving", "0": "sad", "4": "critical"
    },
    "human-annotated": "yes",
    "creation-date": "2020-02-20T21:36:25.000Z01",
    "confidence-map":
    {
      "1": 0.95, "0": 0.77, "4": 0.2
    },
    "type": "groundtruth/text-classification-multilabel"
  }
}
```

## Bounding Box Job Output

The following is sample output (output manifest file) from a bounding box job. For this task, three bounding boxes are returned. The label value contains information about the size of the image, and the location of the bounding boxes.

The `class_id` element is the index of the box's class in the list of available classes for the task. The `class-map` metadata element contains the text of the class.

The metadata has a separate confidence score for each bounding box. The metadata also includes the `class-map` element that maps the `class_id` to the text value of the class. For more information, see [Object Detection Algorithm \(p. 1479\)](#).

The red, italicized text in the examples below depends on labeling job specifications and output data.

```
{
    "source-ref": "s3://AWSDOC-EXAMPLE-BUCKET/example_image.png",
    "bounding-box-attribute-name":
    {
        "image_size": [{ "width": 500, "height": 400, "depth": 3}],
        "annotations":
        [
            {
                "class_id": 0, "left": 111, "top": 134,
                "width": 61, "height": 128},
            {"class_id": 5, "left": 161, "top": 250,
                "width": 30, "height": 30},
            {"class_id": 5, "left": 20, "top": 20,
                "width": 30, "height": 30}
            ]
        },
        "bounding-box-attribute-name-metadata":
        {
            "objects":
            [
                {"confidence": 0.8},
                {"confidence": 0.9},
                {"confidence": 0.9}
            ],
            "class-map":
            {
                "0": "dog",
                "5": "bone"
            },
            "type": "groundtruth/object-detection",
            "human-annotated": "yes",
            "creation-date": "2018-10-18T22:18:13.527256",
            "job-name": "identify-dogs-and-toys"
        }
    }
}
```

The output of a bounding box adjustment job looks like the following JSON. Note that the original JSON is kept intact and two new jobs are listed, each with "adjust-" prepended to the original attribute's name.

```
{
    "source-ref": "S3 bucket location",
    "bounding-box-attribute-name":
    {
        "image_size": [{ "width": 500, "height": 400, "depth": 3}],
        "annotations":
        [
            {
                "class_id": 0, "left": 111, "top": 134,
                "width": 61, "height": 128},
            {"class_id": 5, "left": 161, "top": 250,
                "width": 30, "height": 30},
            {"class_id": 5, "left": 20, "top": 20,
                "width": 30, "height": 30}
            ]
        },
        "bounding-box-attribute-name-metadata":
        {
            "objects":
            [
                {"confidence": 0.8},
                {"confidence": 0.9},
                {"confidence": 0.9}
            ],
            "class-map":
            {
                "0": "dog",
                "5": "bone"
            },
            "type": "groundtruth/object-detection",
            "human-annotated": "yes",
            "creation-date": "2018-10-18T22:18:13.527256",
            "job-name": "adjust-identify-dogs-and-toys"
        }
    }
}
```

```

        "5": "bone"
    },
    "type": "groundtruth/object-detection",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256",
    "job-name": "identify-dogs-and-toys"
},
"adjusted-bounding-box":
{
    "image_size": [{ "width": 500, "height": 400, "depth": 3}],
    "annotations":
    [
        {"class_id": 0, "left": 110, "top": 135,
         "width": 61, "height": 128},
        {"class_id": 5, "left": 161, "top": 250,
         "width": 30, "height": 30},
        {"class_id": 5, "left": 10, "top": 10,
         "width": 30, "height": 30}
    ]
},
"adjusted-bounding-box-metadata":
{
    "objects":
    [
        {"confidence": 0.8},
        {"confidence": 0.9},
        {"confidence": 0.9}
    ],
    "class-map":
    {
        "0": "dog",
        "5": "bone"
    },
    "type": "groundtruth/object-detection",
    "human-annotated": "yes",
    "creation-date": "2018-11-20T22:18:13.527256",
    "job-name": "adjust-bounding-boxes-on-dogs-and-toys",
    "adjustment-status": "adjusted"
}
}
}

```

In this output, the job's type doesn't change, but an adjustment-status field is added. This field has the value of adjusted or unadjusted. If multiple workers have reviewed the object and at least one adjusted the label, the status is adjusted.

## Named Entity Recognition

The following is an example output manifest file from a named entity recognition (NER) labeling task. For this task, seven entities are returned.

In the output manifest, the JSON object, annotations, includes a list of the labels (label categories) that you provided.

Worker responses are in a list named entities. Each entity in this list is a JSON object that contains a label value that matches one in the labels list, an integer startOffset value for labeled span's starting Unicode offset, and an integer endOffset value for the ending Unicode offset.

The metadata has a separate confidence score for each entity. If a single worker labeled each data object, the confidence value for each entity will be zero.

The red, italicized text in the examples below depends on labeling job inputs and worker responses.

```
{
}
```

```

    "source": "Amazon SageMaker is a cloud machine-learning platform that was launched
    in November 2017. SageMaker enables developers to create, train, and deploy machine-
    learning (ML) models in the cloud. SageMaker also enables developers to deploy ML models on
    embedded systems and edge-devices",
    "ner-labeling-job-attribute-name": {
        "annotations": {
            "labels": [
                {
                    "label": "Date",
                    "shortDisplayName": "dt"
                },
                {
                    "label": "Verb",
                    "shortDisplayName": "vb"
                },
                {
                    "label": "Thing",
                    "shortDisplayName": "tng"
                },
                {
                    "label": "People",
                    "shortDisplayName": "ppl"
                }
            ],
            "entities": [
                {
                    "label": "Thing",
                    "startOffset": 22,
                    "endOffset": 53
                },
                {
                    "label": "Thing",
                    "startOffset": 269,
                    "endOffset": 281
                },
                {
                    "label": "Verb",
                    "startOffset": 63,
                    "endOffset": 71
                },
                {
                    "label": "Verb",
                    "startOffset": 228,
                    "endOffset": 234
                },
                {
                    "label": "Date",
                    "startOffset": 75,
                    "endOffset": 88
                },
                {
                    "label": "People",
                    "startOffset": 108,
                    "endOffset": 118
                },
                {
                    "label": "People",
                    "startOffset": 214,
                    "endOffset": 224
                }
            ]
        }
    },
    "ner-labeling-job-attribute-name-metadata": {
        "job-name": "labeling-job/example-ner-labeling-job",
        "type": "groundtruth/text-span",

```

```

"creation-date": "2020-10-29T00:40:39.398470",
"human-annotated": "yes",
"entities": [
    {
        "confidence": 0
    },
    {
        "confidence": 0
    }
]
}

```

## Label Verification Job Output

The output (output manifest file) of a bounding box verification job looks different than the output of a bounding box annotation job. That's because the workers have a different type of task. They're not labeling objects, but evaluating the accuracy of prior labeling, making a judgment, and then providing that judgment and perhaps some comments.

If human workers are verifying or adjusting prior bounding box labels, the output of a verification job would look like the following JSON. The red, italicized text in the examples below depends on labeling job specifications and output data.

```

{
    "source-ref": "s3://AWSDOC-EXAMPLE-BUCKET/image_example.png",
    "bounding-box-attribute-name":
    {
        "image_size": [{ "width": 500, "height": 400, "depth": 3}],
        "annotations":
        [
            {"class_id": 0, "left": 111, "top": 134,
             "width": 61, "height": 128},
            {"class_id": 5, "left": 161, "top": 250,
             "width": 30, "height": 30},
            {"class_id": 5, "left": 20, "top": 20,
             "width": 30, "height": 30}
        ]
    },
    "bounding-box-attribute-name-metadata":
    {
        "objects":
        [
            {"confidence": 0.8},
            {"confidence": 0.9},
            {"confidence": 0.9}
        ],
        "class-map":

```

```
{
    "0": "dog",
    "5": "bone"
},
"type": "groundtruth/object-detection",
"human-annotated": "yes",
"creation-date": "2018-10-18T22:18:13.527256",
"job-name": "identify-dogs-and-toys"
},
"verify-bounding-box-attribute-name": "1",
"verify-bounding-box-attribute-name-metadata":
{
    "class-name": "bad",
    "confidence": 0.93,
    "type": "groundtruth/label-verification",
    "job-name": "verify-bounding-boxes",
    "human-annotated": "yes",
    "creation-date": "2018-11-20T22:18:13.527256",
    "worker-feedback": [
        {"comment": "The bounding box on the bird is too wide on the right side." },
        {"comment": "The bird on the upper right is not labeled." }
    ]
}
}
```

Although the type on the original bounding box output was groundtruth/object-detection, the new type is groundtruth/label-verification. Also note that the worker-feedback array provides worker comments. If the worker doesn't provide comments, the empty fields are excluded during consolidation.

## Semantic Segmentation Job Output

The following is the output manifest file from a semantic segmentation labeling job. The value of the label for this job is a reference to a PNG file in an Amazon S3 bucket.

In addition to the standard elements, the metadata for the label includes a color map that defines which color is used to label the image, the class name associated with the color, and the confidence score for each color. For more information, see [Semantic Segmentation Algorithm \(p. 1502\)](#).

The red, italicized text in the examples below depends on labeling job specifications and output data.

```
{
    "source-ref": "s3://AWSDOC-EXAMPLE-BUCKET/example_city_image.png",
    "city-streets-ref": "S3 bucket location",
    "city-streets-ref-metadata": {
        "internal-color-map": {
            "0": {
                "class-name": "BACKGROUND",
                "confidence": 0.9,
                "hex-color": "#ffffffff"
            },
            "1": {
                "class-name": "buildings",
                "confidence": 0.9,
                "hex-color": "#2acf59"
            },
            "2": {
                "class-name": "road",
                "confidence": 0.9,
                "hex-color": "#f28333"
            }
        }
    },
    "type": "groundtruth/semantic-segmentation",
}
```

```
"human-annotated": "yes",
"creation-date": "2018-10-18T22:18:13.527256",
"job-name": "label-city-streets",
},
"verify-city-streets-ref": "1",
"verify-city-streets-ref-metadata":
{
    "class-name": "bad",
    "confidence": 0.93,
    "type": "groundtruth/label-verification",
    "job-name": "verify-city-streets",
    "human-annotated": "yes",
    "creation-date": "2018-11-20T22:18:13.527256",
    "worker-feedback": [
        {"comment": "The mask on the leftmost building is assigned the wrong side of
the road."},
        {"comment": "The curb of the road is not labeled but the instructions say
otherwise."}
    ]
}
```

Confidence is scored on a per-image basis. Confidence scores are the same across all classes within an image.

The output of a semantic segmentation adjustment job looks similar to the following JSON.

```
{
  "source-ref": "s3://AWSDOC-EXAMPLE-BUCKET/example_city_image.png",
  "city-streets-ref": "S3 bucket location",
  "city-streets-ref-metadata": {
    "internal-color-map": {
      "0": {
        "class-name": "BACKGROUND",
        "confidence": 0.9,
        "hex-color": "#ffffff"
      },
      "1": {
        "class-name": "buildings",
        "confidence": 0.9,
        "hex-color": "#2acf59"
      },
      "2": {
        "class-name": "road",
        "confidence": 0.9,
        "hex-color": "#f28333"
      }
    },
    "type": "groundtruth/semantic-segmentation",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256",
    "job-name": "label-city-streets",
  },
  "adjusted-city-streets-ref": "s3://AWSDOC-EXAMPLE-BUCKET/example_city_image.png",
  "adjusted-city-streets-ref-metadata": {
    "internal-color-map": {
      "0": {
        "class-name": "BACKGROUND",
        "confidence": 0.9,
        "hex-color": "#ffffff"
      },
      "1": {
        "class-name": "buildings",
        "confidence": 0.9,
        "hex-color": "#2acf59"
      }
    }
  }
}
```

```

        "hex-color": "#2acf59"
    },
    "2": {
        "class-name": "road",
        "confidence": 0.9,
        "hex-color": "#f28333"
    }
},
"type": "groundtruth/semantic-segmentation",
"human-annotated": "yes",
"creation-date": "2018-11-20T22:18:13.527256",
"job-name": "adjust-label-city-streets",
}
}

```

## Video Frame Object Detection Output

The following is the output manifest file from a video frame object detection labeling job. The *red, italicized text* in the examples below depends on labeling job specifications and output data.

In addition to the standard elements, the metadata includes a class map that lists each class that has at least one label in the sequence. The metadata also includes job-name which is the name you assigned to the labeling job. For adjustment tasks, If one or more bounding boxes were modified, there is an adjustment-status parameter in the metadata for audit workflows that is set to adjusted.

```

{
    "source-ref": "s3:///example-path/input-manifest.json",
    "CarObjectDetection-ref": "s3://AWSDOC-EXAMPLE-BUCKET/output/labeling-job-name/
annotations/consolidated-annotation/output/0/SeqLabel.json",
    "CarObjectDetection-ref-metadata": {
        "class-map": {
            "0": "car"
            "1": "bus"
        },
        "job-name": "labeling-job/labeling-job-name",
        "human-annotated": "yes",
        "creation-date": "2020-05-15T08:01:16+0000",
        "type": "groundtruth/video-object-detection"
    }
}

```

Ground Truth creates one output sequence file for each sequence of video frames that was labeled. Each output sequence file contains the following:

- All annotations for all frames in a sequence in the detection-annotations list of JSON objects.
- For each frame that was annotated by a worker, the frame file name (frame), number (frame-no), a list of JSON objects containing annotations (annotations), and if applicable, frame-attributes. The name of this list is defined by the task type you use: polylines, polygons, keypoints, and for bounding boxes, annotations.

Each JSON object contains information about a single annotation and associated label. The following table outlines the parameters you'll see for each video frame task type.

Task Type	Parameters
Bounding Box	Box dimensions: height and width Box top, left corner pixel location: top and left

Task Type	Parameters
Keypoint	Keypoint vertices: { "x": int, "y": int }
Polygon	A list of polygon vertices: vertices Polygon vertices: { "x": int, "y": int }  A polygon is a closed shape and so the first point will also represent the last point.
Polyline	A list of polyline vertices: vertices Polyline vertices: { "x": int, "y": int }

In addition to task type specific values, you will see the following in each JSON object:

- Values of any `label-category-attributes` that were specified for that label.
- The `class-id` of the box. Use the `class-map` in the output manifest file to see which label category this ID maps to.

The following is an example of a `SeqLabel.json` file from a bounding box video frame object detection labeling job. This file will be located under `s3://your-output-bucket/output-prefix/annotations/consolidated-annotation/output/annotation-number/`

```
{
    "detection-annotations": [
        {
            "annotations": [
                {
                    "height": 41,
                    "width": 53,
                    "top": 152,
                    "left": 339,
                    "class-id": "1",
                    "label-category-attributes": {
                        "occluded": "no",
                        "size": "medium"
                    }
                },
                {
                    "height": 24,
                    "width": 37,
                    "top": 148,
                    "left": 183,
                    "class-id": "0",
                    "label-category-attributes": {
                        "occluded": "no",
                    }
                }
            ],
            "frame-no": "0",
            "frame": "frame_0000.jpeg",
            "frame-attributes": {name: value, name: value}
        },
        {
            "annotations": [
                {
                    "height": 41,
                    "width": 53,
                    "top": 152,
                    "left": 341,
                    "class-id": "0",
                    "label-category-attributes": {
                        "occluded": "no",
                    }
                }
            ],
            "frame-no": "1",
            "frame": "frame_0001.jpeg",
            "frame-attributes": {name: value, name: value}
        }
    ]
}
```

```

        "label-category-attributes": {}
    },
    {
        "height": 24,
        "width": 37,
        "top": 141,
        "left": 177,
        "class-id": 0,
        "label-category-attributes": {
            "occluded": "no",
        }
    }
],
"frame-no": 1,
"frame": "frame_0001.jpeg",
"frame-attributes": {name: value, name: value}
}
]
}

```

## Video Frame Object Tracking Output

The following is the output manifest file from a video frame object tracking labeling job. The *red, italicized text* in the examples below depends on labeling job specifications and output data.

In addition to the standard elements, the metadata includes a class map that lists each class that has at least one label in the sequence of frames. The metadata also includes job-name which is the name you assigned to the labeling job. For adjustment tasks, if one or more bounding boxes were modified, there is an adjustment-status parameter in the metadata for audit workflows that is set to adjusted.

```

{
    "source-ref": "s3://example-path/input-manifest.json",
    "CarObjectTracking-ref": "s3://AWSDOC-EXAMPLE-BUCKET/output/labeling-job-name/
annotations/consolidated-annotation/output/0/SeqLabel.json",
    "CarObjectTracking-ref-metadata": {
        "class-map": {
            "0": "car"
            "1": "bus"
        },
        "job-name": "labeling-job/labeling-job-name",
        "human-annotated": "yes",
        "creation-date": "2020-05-15T08:01:16+0000",
        "type": "groundtruth/video-object-tracking"
    }
}

```

Ground Truth creates one output sequence file for each sequence of video frames that was labeled. Each output sequence file contains the following:

- All annotations for all frames in a sequence in the tracking-annotations list of JSON objects.
- For each frame that was annotated by a worker, the frame (frame), number (frame-no), a list of JSON objects containing annotations (annotations), and if applicable, frame attributes (frame-attributes). The name of this list is defined by the task type you use: polylines, polygons, keypoints, and for bounding boxes, annotations.

Each JSON object contains information about a single annotation and associated label. The following table outlines the parameters you'll see for each video frame task type.

Task Type	Parameters
Bounding Box	Box dimensions: <code>height</code> and <code>width</code> Box top, left corner pixel location: <code>top</code> and <code>left</code>
Keypoint	Keypoint vertices: { "x": int, "y": int }
Polygon	A list of polygon vertices: <code>vertices</code> Polygon vertices: { "x": int, "y": int }  A polygon is a closed shape and so the first point will also represent the last point.
Polyline	A list of polyline vertices: <code>vertices</code> Polyline vertices: { "x": int, "y": int }

In addition to task type specific values, you will see the following in each JSON object:

- Values of any `label-category-attributes` that were specified for that label.
- The `class-id` of the box. Use the `class-map` in the output manifest file to see which label category this ID maps to.
- An `object-id` which identifies an instance of a label. This ID will be the same across frames if a worker identified the same instance of an object in multiple frames. For example, if a car appeared in multiple frames, all bounding boxes uses to identify that car would have the same `object-id`.
- The `object-name` which is the instance ID of that annotation.

The following is an example of a `SeqLabel.json` file from a bounding box video frame object tracking labeling job. This file will be located under `s3://your-output-bucket/output-prefix/annotations/consolidated-annotation/output/annotation-number/`

```
{
  "tracking-annotations": [
    {
      "annotations": [
        {
          "height": 36,
          "width": 46,
          "top": 178,
          "left": 315,
          "class-id": "0",
          "label-category-attributes": {
            "occluded": "no"
          },
          "object-id": "480dc450-c0ca-11ea-961f-a9b1c5c97972",
          "object-name": "car:1"
        }
      ],
      "frame-no": "0",
      "frame": "frame_0001.jpeg",
      "frame-attributes": {}
    },
    {
      "annotations": [
        {
          "height": 30,
          "width": 47,
          "top": 163,
          "left": 315
        }
      ],
      "frame-no": "1",
      "frame": "frame_0002.jpeg",
      "frame-attributes": {}
    }
  ]
}
```

```

        "left": 344,
        "class-id": "1",
        "label-category-attributes": {
            "occluded": "no",
            "size": "medium"
        },
        "object-id": "98f2b0b0-c0ca-11ea-961f-a9b1c5c97972",
        "object-name": "bus:1"
    },
    {
        "height": 28,
        "width": 33,
        "top": 150,
        "left": 192,
        "class-id": "0",
        "label-category-attributes": {
            "occluded": "partially"
        },
        "object-id": "480dc450-c0ca-11ea-961f-a9b1c5c97972",
        "object-name": "car:1"
    }
],
"frame-no": "1",
"frame": "frame_0002.jpeg",
"frame-attributes": {name: value, name: value}
}
]
}

```

## 3D Point Cloud Semantic Segmentation Output

The following is the output manifest file from a 3D point cloud semantic segmentation labeling job.

In addition to the standard elements, the metadata for the label includes a color map that defines which color is used to label the image, the class name associated with the color, and the confidence score for each color. Additionally, there is an adjustment-status parameter in the metadata for audit workflows that is set to adjusted if the color mask is modified. If you added one or more frameAttributes to your label category configuration file, worker responses for frame attributes are in the JSON object, dataset-object-attributes.

The `your-label-attribute-ref` parameter contains the location of a compressed file with a .zlib extension. When you uncompress this file, it contains an array. Each index in the array corresponds to the index of an annotated point in the input point cloud. The value of the array at a given index gives the class of the point at the same index in the point cloud, based on the semantic color map found in the `color-map` parameter of the metadata.

You can use Python code similar to the following to decompress a .zlib file:

```

import zlib
from array import array

# read the label file
compressed_binary_file = open(zlib_file_path/file.zlib, 'rb').read()

# uncompress the label file
binary_content = zlib.decompress(compressed_binary_file)

# load labels to an array
my_int_array_data = array('B', binary_content);

print(my_int_array_data)

```

The code block above will produce an output similar to the following. Each element of the printed array contains the class of a point at the that index in the point cloud. For example, `my_int_array_data[0] = 1` means `point[0]` in the input point cloud has a class 1. In the following output manifest file example, class 0 corresponds with "Background", 1 with Car, and 2 with Pedestrian.

The following is an example of a semantic segmentation 3D point cloud labeling job output manifest file. The red, italicized text in the examples below depends on labeling job specifications and output data.

```

{
  "source-ref": "s3://AWSDOC-EXAMPLE-BUCKET/examplefolder/frame1.bin",
  "source-ref-metadata": {
    "format": "binary/xyzzi",
    "unix-timestamp": 1566861644.759115,
    "ego-vehicle-pose": {...},
    "prefix": "s3://AWSDOC-EXAMPLE-BUCKET/lidar_singleframe_dataset/prefix",
    "images": [{...}]
  },
  "lidar-ss-label-attribute-ref": "s3://your-output-bucket/labeling-job-name/annotations/consolidated-annotation/output/dataset-object-id/filename.zlib",
  "lidar-ss-label-attribute-ref-metadata": {
    'color-map': {
      "0": {
        "class-name": "Background",
        "hex-color": "#ffffff",
        "confidence": 0.00
      },
      "1": {
        "class-name": "Car",
        "hex-color": "#2ca02c",
        "confidence": 0.00
      },
      "2": {
        "class-name": "Pedestrian",
        "hex-color": "#1f77b4",
        "confidence": 0.00
      },
      "3": {
        "class-name": "Tree",
        "hex-color": "#ff7f0e",
        "confidence": 0.00
      }
    },
    'type': 'groundtruth/point_cloud_single_frame_semantic_segmentation',
    'human-annotated': 'yes',
    'creation-date': '2019-11-12T01:18:14.271944',
    'job-name': 'labeling-job-name',
    //only present for adjustment audit workflow
    "adjustment-status": "adjusted", // "adjusted" means the label was adjusted
    "dataset-object-attributes": {name: value, name: value}
  }
}

```

## 3D Point Cloud Object Detection Output

The following is sample output from a 3D point cloud object detection job. For this task type, the data about 3D cuboids is returned in the `3d-bounding-box` parameter, in a list named `annotations`. In this list, each 3D cuboid is described using the following information.

- Each class, or label category, that you specify in your input manifest is associated with a `class-id`. Use the `class-map` to identify the class associated with each class ID.
- These classes are used to give each 3D cuboid an `object-name` in the format `<class>:<integer>` where `integer` is a unique number to identify that cuboid in the frame.
- `center-x`, `center-y`, and `center-z` are the coordinates of the center of the cuboid, in the same coordinate system as the 3D point cloud input data used in your labeling job.
- `length`, `width`, and `height` describe the dimensions of the cuboid.
- `yaw` is used to describe the orientation (heading) of the cuboid in radians.

The `yaw` measurement in the output data is 180 degrees, or  $\pi$  in radians, minus `yaw` in the right handed world coordinate system when looking down at the cuboid. In other words, when looking at a cuboid from the top-down, `yaw_in_output_data` is clockwise-positive (in contrast to the right handed world coordinate system, in which the top-down view is associated with counter-clockwise-positive rotation). When looking up from the cuboid, `yaw_in_output_data` is counterclockwise-positive.

To convert `yaw_in_output_data` to the more common orientation of the right handed world coordinate system, use the following (all units are in radians):

```
yaw_right_handed_cartesian_system = pi - yaw_in_output_data
```

- If you created a 3D point cloud adjustment labeling job and included `pitch` and `roll` in the input manifest file, the same `pitch` and `roll` measurements will appear in the output manifest file. Otherwise, `pitch` and `roll` will always be 0.
- If you included label attributes in your input manifest file for a given class, a `label-category-attributes` parameter is included for all cuboids for which workers selected label attributes.

If one or more cuboids were modified, there is an `adjustment-status` parameter in the metadata for audit workflows that is set to `adjusted`. If you added one or more `frameAttributes` to your label category configuration file, worker responses for frame attributes are in the JSON object, `dataset-object-attributes`.

The *red, italicized text* in the examples below depends on labeling job specifications and output data. The ellipses (...) denote a continuation of that list, where additional objects with the same format as the proceeding object can appear.

```
{
  "source-ref": "s3://AWSDOC-EXAMPLE-BUCKET/examplefolder/frame1.txt",
  "source-ref-metadata": {
    "format": "text/xyz",
    "unix-timestamp": 1566861644.759115,
    "prefix": "s3://AWSDOC-EXAMPLE-BUCKET/lidar_singleframe_dataset/prefix",
    "ego-vehicle-pose": {
      "heading": {
        "qx": -0.0211296123795955,
        "qy": -0.006495469416730261,
        "qz": -0.008024565904865688,
        "qw": 0.9997181192298087
      },
      "position": {
        "x": -2.7161461413869947,
        "y": 116.25822288149078,
        "z": 1.8348751887989483
      }
    },
    "images": [
      {
        "fx": 847.7962624528487,
        "fy": 500.0,
        "tx": 0.0,
        "ty": 0.0
      }
    ]
}
```

```

        "fy": 850.0340893791985,
        "cx": 576.2129134707038,
        "cy": 317.2423573573745,
        "k1": 0,
        "k2": 0,
        "k3": 0,
        "k4": 0,
        "p1": 0,
        "p2": 0,
        "skew": 0,
        "unix-timestamp": 1566861644.759115,
        "image-path": "images/frame_0_camera_0.jpg",
        "position": {
            "x": -2.2722515189268138,
            "y": 116.86003310568965,
            "z": 1.454614668542299
        },
        "heading": {
            "qx": 0.7594754093069037,
            "qy": 0.02181790885672969,
            "qz": -0.02461725233103356,
            "qw": -0.6496916273040025
        },
        "camera_model": "pinhole"
    }
],
},
"3d-bounding-box":
{
    "annotations": [
        {
            "label-category-attributes": {
                "Occlusion": "Partial",
                "Type": "Sedan"
            },
            "object-name": "Car:1",
            "class-id": 0,
            "center-x": -2.616382013657516,
            "center-y": 125.04149850484193,
            "center-z": 0.311272296465834,
            "length": 2.993000265181146,
            "width": 1.8355260519692056,
            "height": 1.3233490884304047,
            "roll": 0,
            "pitch": 0,
            "yaw": 1.6479308313703527
        },
        {
            "label-category-attributes": {
                "Occlusion": "Partial",
                "Type": "Sedan"
            },
            "object-name": "Car:2",
            "class-id": 0,
            "center-x": -5.188984560617168,
            "center-y": 99.7954483288783,
            "center-z": 0.2226435567445657,
            "length": 4,
            "width": 2,
            "height": 2,
            "roll": 0,
            "pitch": 0,
            "yaw": 1.6243170732068055
        }
    ]
},
}

```

```

"3d-bounding-box-metadata":
{
    "objects": [],
    "class_map":
    {
        "0": "Car",
    },
    "type": "groundtruth/point_cloud_object_detection",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256",
    "job-name": "identify-3d-objects",
    "adjustment-status": "adjusted",
    "dataset-object-attributes": {name: value, name: value}
}
}

```

## 3D Point Cloud Object Tracking Output

The following is an example of an output manifest file from a 3D point cloud object tracking labeling job. The *red, italicized text* in the examples below depends on labeling job specifications and output data. The ellipses (...) denote a continuation of that list, where additional objects with the same format as the proceeding object can appear.

In addition to the standard elements, the metadata includes a class map that lists each class that has at least one label in the sequence. If one or more cuboids were modified, there is an `adjustment-status` parameter in the metadata for audit workflows that is set to `adjusted`.

```

{
    "source-ref": "s3://AWSDOC-EXAMPLE-BUCKET/myfolder/seq1.json",
    "lidar-label-attribute-ref": "s3://<CustomerOutputLocation>/<labelingJobName>/annotations/consolidated-annotation/output/<datasetObjectId>/SeqLabel.json",
    "lidar-label-attribute-ref-metadata": {
        "objects": [
            [
                {
                    "frame-no": 300,
                    "confidence": []
                },
                {
                    "frame-no": 301,
                    "confidence": []
                },
                ...
            ],
            "class-map": {'0': 'Car', '1': 'Person'},
            "type": 'groundtruth/point_cloud_object_tracking',
            "human-annotated": 'yes',
            "creation-date": '2019-11-12T01:18:14.271944',
            "job-name": 'identify-3d-objects',
            "adjustment-status": "adjusted"
        }
    }
}

```

In the above example, the cuboid data for each frame in `seq1.json` is in `SeqLabel.json` in the Amazon S3 location, `s3://<customerOutputLocation>/<labelingJobName>/annotations/consolidated-annotation/output/<datasetObjectId>/SeqLabel.json`. The following is an example of this label sequence file.

For each frame in the sequence, you see the `frame-number`, `frame-name`, if applicable, `frame-attributes`, and a list of annotations. This list contains 3D cuboids that were drawn for that frame. Each annotation includes the following information:

- An object-name in the format <class>:<integer> where class identifies the label category and integer is a unique ID across the dataset.
- When workers draw a cuboid, it is associated with a unique object-id which is associated with all cuboids that identify the same object across multiple frames.
- Each class, or label category, that you specified in your input manifest is associated with a class-id. Use the class-map to identify the class associated with each class ID.
- center-x, center-y, and center-z are the coordinates of the center of the cuboid, in the same coordinate system as the 3D point cloud input data used in your labeling job.
- length, width, and height describe the dimensions of the cuboid.
- yaw is used to describe the orientation (heading) of the cuboid in radians.

The yaw measurement in the output data is 180 degrees, or pi in radians, minus yaw in the right handed world coordinate system when looking down at the cuboid. In other words, when looking at a cuboid from the top-down, yaw\_in\_output\_data is clockwise-positive (in contrast to the right handed world coordinate system, in which the top-down view is associated with counter-clockwise-positive rotation). When looking up from the cuboid, yaw\_in\_output\_data is counterclockwise-positive.

To convert yaw\_in\_output\_data to the more common orientation of the right handed world coordinate system, use the following (all units are in radians):

```
yaw_right_handed_cartesian_system = pi - yaw_in_output_data
```

- If you created a 3D point cloud adjustment labeling job and included pitch and roll in the input manifest file, the same pitch and roll measurements will appear in the output manifest file. Otherwise, pitch and role will always be 0.
- If you included label attributes in your input manifest file for a given class, a label-category-attributes parameter is included for all cuboids for which workers selected label attributes.

```
{
    "tracking-annotations": [
        {
            "frame-number": 0,
            "frame-name": "0.txt.pcd",
            "frame-attributes": {name: value, name: value},
            "annotations": [
                {
                    "label-category-attributes": {},
                    "object-name": "Car:4",
                    "class-id": 0,
                    "center-x": -2.2906369208300674,
                    "center-y": 103.73924823843463,
                    "center-z": 0.37634114027023313,
                    "length": 4,
                    "width": 2,
                    "height": 2,
                    "roll": 0,
                    "pitch": 0,
                    "yaw": 1.5827222214406014,
                    "object-id": "ae5dc770-a782-11ea-b57d-67c51a0561a1"
                },
                {
                    "label-category-attributes": {
                        "Occlusion": "Partial",
                        "Type": "Sedan"
                    },
                    "object-name": "Car:1",
                    "class-id": 0,
                    "center-x": -2.2906369208300674,
                    "center-y": 103.73924823843463,
                    "center-z": 0.37634114027023313,
                    "length": 4,
                    "width": 2,
                    "height": 2,
                    "roll": 0,
                    "pitch": 0,
                    "yaw": 1.5827222214406014,
                    "object-id": "ae5dc770-a782-11ea-b57d-67c51a0561a1"
                }
            ]
        }
    ]
}
```

```

    "center-x": -2.6451293634707413,
    "center-y": 124.9534455706848,
    "center-z": 0.5020834081743839,
    "length": 4,
    "width": 2,
    "height": 2.080488827301309,
    "roll": 0,
    "pitch": 0,
    "yaw": -1.5963335581398077,
    "object-id": "06efb020-a782-11ea-b57d-67c51a0561a1"
},
{
    "label-category-attributes": {
        "Occlusion": "Partial",
        "Type": "Sedan"
    },
    "object-name": "Car:2",
    "class-id": 0,
    "center-x": -5.205611313118477,
    "center-y": 99.91731932137061,
    "center-z": 0.22917217081212138,
    "length": 3.8747142207671956,
    "width": 1.9999999999999918,
    "height": 2,
    "roll": 0,
    "pitch": 0,
    "yaw": 1.5672228760316775,
    "object-id": "26fad020-a782-11ea-b57d-67c51a0561a1"
}
],
},
{
    "frame-number": 1,
    "frame-name": "1.txt.pcd",
    "frame-attributes": {},
    "annotations": [
        {
            "label-category-attributes": {},
            "object-name": "Car:4",
            "class-id": 0,
            "center-x": -2.2906369208300674,
            "center-y": 103.73924823843463,
            "center-z": 0.37634114027023313,
            "length": 4,
            "width": 2,
            "height": 2,
            "roll": 0,
            "pitch": 0,
            "yaw": 1.582722214406014,
            "object-id": "ae5dc770-a782-11ea-b57d-67c51a0561a1"
        },
        {
            "label-category-attributes": {
                "Occlusion": "Partial",
                "Type": "Sedan"
            },
            "object-name": "Car:1",
            "class-id": 0,
            "center-x": -2.6451293634707413,
            "center-y": 124.9534455706848,
            "center-z": 0.5020834081743839,
            "length": 4,
            "width": 2,
            "height": 2.080488827301309,
            "roll": 0,
            "pitch": 0,
            "yaw": -1.5963335581398077,
            "object-id": "06efb020-a782-11ea-b57d-67c51a0561a1"
        }
    ]
}

```

```
        "yaw": -1.5963335581398077,
        "object-id": "06efb020-a782-11ea-b57d-67c51a0561a1"
    },
    {
        "label-category-attributes": {
            "Occlusion": "Partial",
            "Type": "Sedan"
        },
        "object-name": "Car:2",
        "class-id": 0,
        "center-x": -5.221311072916759,
        "center-y": 100.4639841045424,
        "center-z": 0.22917217081212138,
        "length": 3.8747142207671956,
        "width": 1.9999999999999918,
        "height": 2,
        "roll": 0,
        "pitch": 0,
        "yaw": 1.5672228760316775,
        "object-id": "26fad020-a782-11ea-b57d-67c51a0561a1"
    }
}
]
```

## Enhanced Data Labeling

Amazon SageMaker Ground Truth manages sending your data objects to workers to be labeled. Labeling each data object is a *task*. Workers complete each task until the entire labeling job is complete. Ground Truth divides the total number of tasks into smaller *batches* that are sent to workers. A new batch is sent to workers when the previous one is finished.

Ground Truth provides two features that help improve the accuracy of your data labels and reduce the total cost of labeling your data:

- *Annotation consolidation* helps to improve the accuracy of your data object labels. It combines the results of multiple workers' annotation tasks into one high-fidelity label.
- *Automated data labeling* uses machine learning to label portions of your data automatically without having to send them to human workers.

### Topics

- [Control the Flow of Data Objects Sent to Workers \(p. 427\)](#)
- [Consolidate Annotations \(p. 429\)](#)
- [Automate Data Labeling \(p. 430\)](#)
- [Chaining Labeling Jobs \(p. 436\)](#)

## Control the Flow of Data Objects Sent to Workers

Depending on the type of labeling job you create, Amazon SageMaker Ground Truth sends data objects to workers in batches or in a streaming fashion. You can control the flow of data objects to workers in the following ways:

- For both types of labeling jobs, you can use `MaxConcurrentTaskCount` to control the total number of data objects available to all workers at a given point in time when the labeling job is running.
- For streaming labeling jobs, you can control the flow of data objects to workers by monitoring and controlling the number of data objects sent to the Amazon SQS associated with your labeling job.

Use the following sections to learn more about these options. To learn more about streaming labeling jobs, see [Ground Truth Streaming Labeling Jobs \(p. 366\)](#).

### Topics

- [Use MaxConcurrentTaskCount to Control the Flow of Data Objects \(p. 428\)](#)
- [Use Amazon SQS to Control the Flow of Data Objects to Streaming Labeling Jobs \(p. 428\)](#)

## Use MaxConcurrentTaskCount to Control the Flow of Data Objects

`MaxConcurrentTaskCount` defines the maximum number of data objects that can be labeled by human workers at the same time. If you use the console, this parameter is set to 1,000. If you use `CreateLabelingJob`, you can set this parameter to any integer between 1 and 1,000, inclusive.

When you start a labeling job using an input manifest file, Ground Truth does the following:

1. For each data object listed in your input manifest file, one or more tasks are created, depending on the value you specify for `NumberOfHumanWorkersPerDataObject`. For example, if you set the number of workers per data object to 3, 3 tasks will be created for each dataset object. To be marked as successfully labeled, at least one worker must label the object. Alternatively, the tasks can expire or be declined.
2. If you are using the Mechanical Turk workforce, Ground Truth first sends a batch of 10 dataset objects to your workers. It uses this small batch to set up the labeling job and to make sure that the job is correctly configured.
3. Next, Ground Truth sends `MaxConcurrentTaskCount` number of dataset objects to workers. For example, if you have 2,000 input data objects in your input manifest file and have set the number of workers per data object to 3 and set `MaxConcurrentTaskCount` to 900, the first 900 data objects in your input manifest are sent to workers, corresponding to 2,700 tasks (900 x 3). This is the first full-sized set of objects sent to workers.
4. What happens next depends on the type of labeling job you create. This step assumes one or more dataset objects in your input manifest file, or sent using an Amazon SNS input data source (in a streaming labeling job) were not included in the set sent to workers in step 3.
  - **Streaming labeling job:** As long as the total number of objects available to workers is equal to `MaxConcurrentTaskCount`, all remaining dataset objects on your input manifest file and that you send in real time using Amazon SNS are placed on an Amazon SQS queue. When the total number of objects available to workers falls below `MaxConcurrentTaskCount` minus `NumberOfHumanWorkersPerDataObject`, a new data object from the queue is used to create `NumberOfHumanWorkersPerDataObject`-tasks, which are sent to workers in real time.
  - **Non-streaming labeling job:** As workers finish labeling one set of objects, up to `MaxConcurrentTaskCount` times `NumberOfHumanWorkersPerDataObject` number of new tasks will be sent to workers. This process is repeated until all data objects in the input manifest file are labeled.

## Use Amazon SQS to Control the Flow of Data Objects to Streaming Labeling Jobs

When you create a streaming labeling job, an Amazon SQS queue is automatically created in your account. Data objects are only added to the Amazon SQS queue when the total number of objects sent to workers is above `MaxConcurrentTaskCount`. Otherwise, objects are sent directly to workers.

You can use this queue to manage the flow of data objects to your labeling job. To learn more, see [Manage Labeling Requests with an Amazon SQS Queue \(p. 369\)](#).

## Consolidate Annotations

An *annotation* is the result of a single worker's labeling task. *Annotation consolidation* combines the annotations of two or more workers into a single label for your data objects. A label, which is assigned to each object in the dataset, is a probabilistic estimate of what the true label should be. Each object in the dataset typically has multiple annotations, but only one label or set of labels.

You decide how many workers annotate each object in your dataset. Using more workers can increase the accuracy of your labels, but also increases the cost of labeling. To learn more about Ground Truth pricing, see [Amazon SageMaker Ground Truth pricing](#).

If you use the Amazon SageMaker console to create a labeling job, the following are the defaults for the number of workers who can annotate objects:

- Text classification—3 workers
- Image classification—3 workers
- Bounding boxes—5 workers
- Semantic segmentation—3 workers
- Named entity recognition—3 workers

When you use the [CreateLabelingJob](#) operation, you set the number of workers to annotate each data object with the `NumberOfHumanWorkersPerDataObject` parameter. You can override the default number of workers that annotate a data object using the console or the [CreateLabelingJob](#) operation.

Ground Truth provides an annotation consolidation function for each of its predefined labeling tasks: bounding box, image classification, name entity recognition, semantic segmentation, and text classification. These are the functions:

- Multi-class annotation consolidation for image and text classification uses a variant of the [Expectation Maximization](#) approach to annotations. It estimates parameters for each worker and uses Bayesian inference to estimate the true class based on the class annotations from individual workers.
- Bounding box annotation consolidates bounding boxes from multiple workers. This function finds the most similar boxes from different workers based on the [Jaccard index](#), or intersection over union, of the boxes and averages them.
- Semantic segmentation annotation consolidation treats each pixel in a single image as a multi-class classification. This function treats the pixel annotations from workers as "votes," with more information from surrounding pixels incorporated by applying a smoothing function to the image.
- Named entity recognition clusters text selections by Jaccard similarity and calculates selection boundaries based on the mode, or the median if the mode isn't clear. The label resolves to the most assigned entity label in the cluster, breaking ties by random selection.

You can use other algorithms to consolidate annotations. For information, see [Create Your Own Annotation Consolidation Function \(p. 429\)](#).

### Create Your Own Annotation Consolidation Function

You can choose to use your own annotation consolidation function to determine the final labels for your labeled objects. There are many possible approaches for writing a function and the approach that you take depends on the nature of the annotations to consolidate. Broadly, consolidation functions look at the annotations from workers, measure the similarity between them, and then use some form of probabilistic judgment to determine what the most probable label should be.

If you want to use other algorithms to create annotation consolidations functions, you can find the worker responses in the `[project-name]/annotations/worker-response` folder of the Amazon S3 bucket where you direct the job output.

## Assess Similarity

To assess the similarity between labels, you can use one of the following strategies, or you can use one that meets your data labeling needs:

- For label spaces that consist of discrete, mutually exclusive categories, such as multi-class classification, assessing similarity can be straightforward. Discrete labels either match or do not match.
- For label spaces that don't have discrete values, such as bounding box annotations, find a broad measure of similarity. For bounding boxes, one such measure is the Jaccard index. This measures the ratio of the intersection of two boxes with the union of the boxes to assess how similar they are. For example, if there are three annotations, then there can be a function that determines which annotations represent the same object and should be consolidated.

## Assess the Most Probable Label

With one of the strategies detailed in the previous sections in mind, make some sort of probabilistic judgment on what the consolidated label should be. In the case of discrete, mutually exclusive categories, this can be straightforward. One of the most common ways to do this is to take the results of a majority vote between the annotations. This weights the annotations equally.

Some approaches attempt to estimate the accuracy of different annotators and weight their annotations in proportion to the probability of correctness. An example of this is the Expectation Maximization method, which is used in the default Ground Truth consolidation function for multi-class annotations.

For more information about creating an annotation consolidation function, see [Step 3: Processing with Amazon Lambda \(p. 307\)](#).

## Automate Data Labeling

If you choose, Amazon SageMaker Ground Truth can use active learning to automate the labeling of your input data for certain built-in task types. *Active learning* is a machine learning technique that identifies data that should be labeled by your workers. In Ground Truth, this functionality is called automated data labeling. Automated data labeling helps to reduce the cost and time that it takes to label your dataset compared to using only humans. When you use automated labeling, you incur SageMaker training and inference costs.

We recommend using automated data labeling on large datasets because the neural networks used with active learning require a significant amount of data for every new dataset. Typically, as you provide more data, the potential for high accuracy predictions goes up. Data will only be auto-labeled if the neural network used in the auto-labeling model can achieve an acceptably high level of accuracy. Therefore, with larger datasets, there is more potential to automatically label the data because the neural network can achieve high enough accuracy for auto-labeling. Automated data labeling is most appropriate when you have thousands of data objects. The minimum number of objects allowed for automated data labeling is 1,250, but we strongly suggest providing a minimum of 5,000 objects.

Automated data labeling is available only for the following Ground Truth built-in task types:

- [Image Classification \(Single Label\) \(p. 180\)](#)
- [Image Semantic Segmentation \(p. 177\)](#)
- [Object detection \(Bounding Box \(p. 171\)\)](#)
- [Text Classification \(Single Label\) \(p. 191\)](#)

[Streaming labeling jobs](#) do not support automated data labeling.

To learn how to create a custom active learning workflow using your own model, see [Set up an active learning workflow with your own model \(p. 436\)](#).

Input data quotas apply for automated data labeling jobs. See [Input Data Quotas \(p. 371\)](#) for information about dataset size, input data size and resolution limits.

**Note**

Before you use an the automated-labeling model in production, you need to fine-tune or test it, or both. You might fine-tune the model (or create and tune another supervised model of your choice) on the dataset produced by your labeling job to optimize the model's architecture and hyperparameters. If you decide to use the model for inference without fine-tuning it, we strongly recommend making sure that you evaluate its accuracy on a representative (for example, randomly selected) subset of the dataset labeled with Ground Truth and that it matches your expectations.

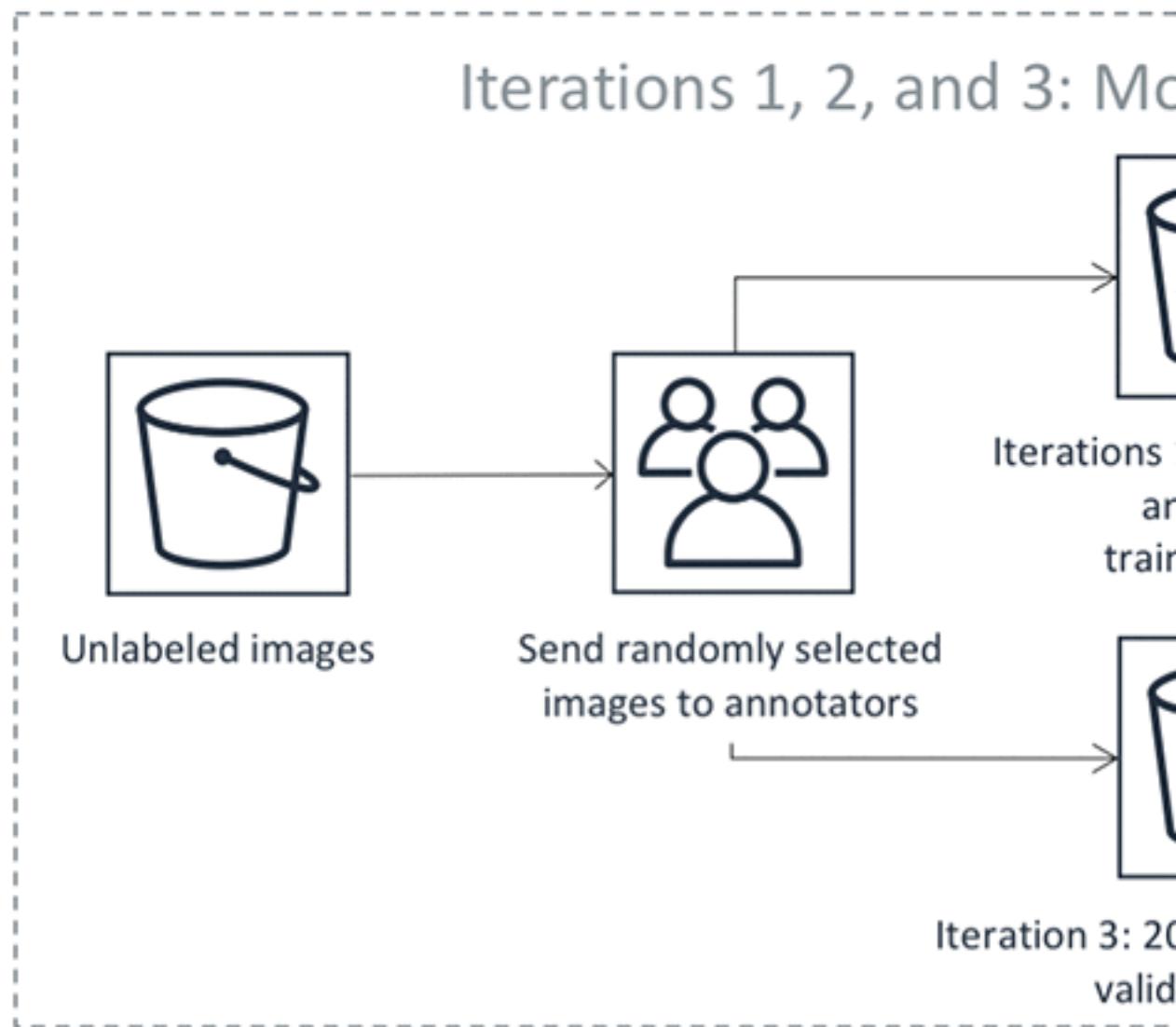
## How it Works

You enable automated data labeling when you create a labeling job. This is how it works:

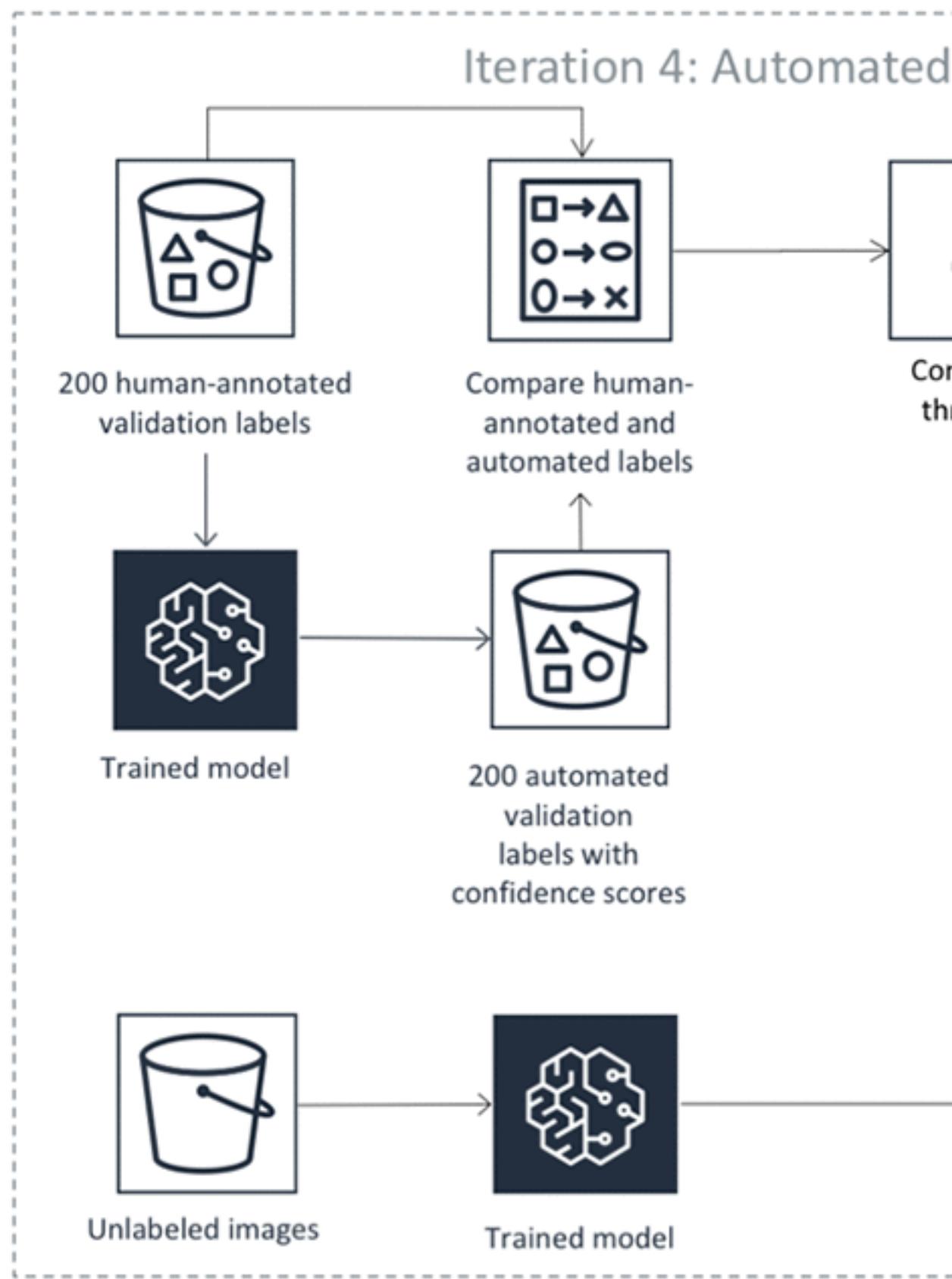
1. When Ground Truth starts an automated data labeling job, it selects a random sample of input data objects and sends them to human workers. If more than 10% of these data objects fail, the labeling job will fail. If the labeling job fails, in addition to reviewing any error message Ground Truth returns, check that your input data is displaying correctly in the worker UI, instructions are clear, and that you have given workers enough time to complete tasks.
2. When the labeled data is returned, it is used to create a training set and a validation set. Ground Truth uses these datasets to train and validate the model used for auto-labeling.
3. Ground Truth runs a batch transform job, using the validated model for inference on the validation data. Batch inference produces a confidence score and quality metric for each object in the validation data.
4. The auto labeling component will use these quality metrics and confidence scores to create a *confidence score threshold* that ensures quality labels.
5. Ground Truth runs a batch transform job on the unlabeled data in the dataset, using the same validated model for inference. This produces a confidence score for each object.
6. The Ground Truth auto labeling component determines if the confidence score produced in step 5 for each object meets the required threshold determined in step 4. If the confidence score meets the threshold, the expected quality of automatically labeling exceeds the requested level of accuracy and that object is considered auto-labeled.
7. Step 6 produces a dataset of unlabeled data with confidence scores. Ground Truth selects data points with low confidence scores from this dataset and sends them to human workers.
8. Ground Truth uses the existing human-labeled data and this additional labeled data from human workers to update the model.
9. The process is repeated until the dataset is fully labeled or until another stopping condition is met. For example, auto-labeling stops if your human annotation budget is reached.

The preceding steps happen in iterations. Select each tab in the following table to see an example of the processes that happen in each iteration for an object detection automated labeling job. The number of data objects used in a given step in these images (for example, 200) is specific to this example. If there are fewer than 5,000 objects to label, the validation set size is 20% of the whole dataset. If there are more than 5,000 objects in your input dataset, the validation set size is 10% of the whole dataset. You can control the number of human labels collected per active learning iteration by changing the value for `MaxConcurrentTaskCount` when using the API operation `CreateLabelingJob`. This value is set to 1,000 when you create a labeling job using the console. In the active learning flow illustrated under the **Active Learning** tab, this value is set to 200.

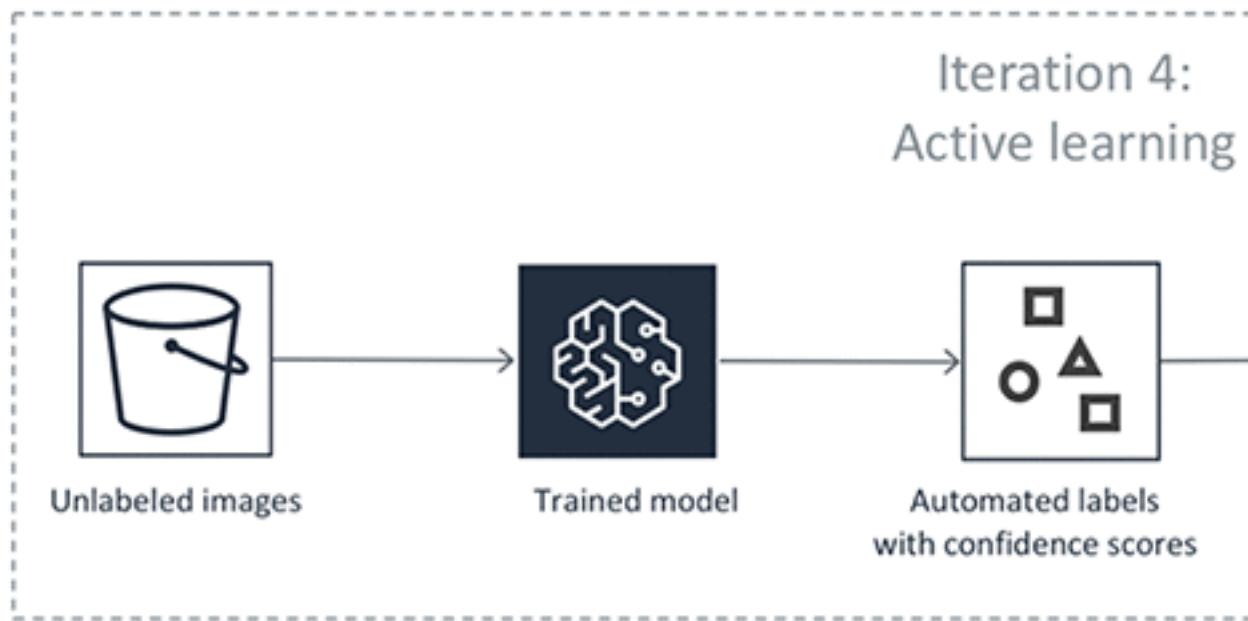
Model Training



Automated Labeling



## Active Learning



### Accuracy of Automated Labels

The definition of *accuracy* depends on the built-in task type that you use with automated labeling. For all task types, these accuracy requirements are pre-determined by Ground Truth and cannot be manually configured.

- For image classification and text classification, Ground Truth uses logic to find a label-prediction confidence level that corresponds to at least 95% label accuracy. This means Ground Truth expects the accuracy of the automated labels to be at least 95% when compared to the labels that human labelers would provide for those examples.
- For bounding boxes, the expected mean [Intersection Over Union \(IoU\)](#) of the auto-labeled images is 0.6. To find the mean IoU, Ground Truth calculates the mean IoU of all the predicted and missed boxes on the image for every class, and then averages these values across classes.
- For semantic segmentation, the expected mean IoU of the auto-labeled images is 0.7. To find the mean IoU, Ground Truth takes the mean of the IoU values of all the classes in the image (excluding the background).

At every iteration of Active Learning (steps 3-6 in the list above), the confidence threshold is found using the human-annotated validation set so that the expected accuracy of the auto-labeled objects satisfies certain predefined accuracy requirements.

### Create an Automated Data Labeling Job (Console)

To create a labeling job that uses automated labeling in the SageMaker console, use the following procedure.

#### To create an automated data labeling job (console)

1. Open the Ground Truth **Labeling jobs** section of the SageMaker console: <https://console.amazonaws.cn/sagemaker/groundtruth>.
2. Using [Create a Labeling Job \(Console\) \(p. 336\)](#) as a guide, complete the **Job overview** and **Task type** sections. Note that auto labeling is not supported for custom task types.

3. Under **Workers**, choose your workforce type.
4. In the same section, choose **Enable automated data labeling**.
5. Using [Step 4: Configure the Bounding Box Tool \(p. 170\)](#) as a guide, create worker instructions in the section **Task Type labeling tool**. For example, if you chose **Semantic segmentation** as your labeling job type, this section is called **Semantic segmentation labeling tool**.
6. To preview your worker instructions and dashboard, choose **Preview**.
7. Choose **Create**. This creates and starts your labeling job and the auto labeling process.

You can see your labeling job appear in the **Labeling jobs** section of the SageMaker console. Your output data appears in the Amazon S3 bucket that you specified when creating the labeling job. For more information about the format and file structure of your labeling job output data, see [Output Data \(p. 404\)](#).

## Create an Automated Data Labeling Job (API)

To create an automated data labeling job using the SageMaker API, use the `LabelingJobAlgorithmsConfig` parameter of the `CreateLabelingJob` operation. To learn how to start a labeling job using the `CreateLabelingJob` operation, see [Create a Labeling Job \(API\) \(p. 338\)](#).

Specify the Amazon Resource Name (ARN) of the algorithm that you are using for automated data labeling in the `LabelingJobAlgorithmSpecificationArn` parameter. Choose from one of the four Ground Truth built-in algorithms that are supported with automated labeling:

- [Image Classification \(Single Label\) \(p. 180\)](#)
- [Image Semantic Segmentation \(p. 177\)](#)
- [Object detection \(Bounding Box \(p. 171\)\)](#)
- [Text Classification \(Single Label\) \(p. 191\)](#)

When an automated data labeling job finishes, Ground Truth returns the ARN of the model it used for the automated data labeling job. Use this model as the starting model for similar auto-labeling job types by providing the ARN, in string format, in the `InitialActiveLearningModelArn` parameter. To retrieve the model's ARN, use an Amazon Command Line Interface (Amazon CLI) command similar to the following.

```
# Fetch the mARN of the model trained in the final iteration of the previous labeling
# job.Ground Truth
pretrained_model_arn = sagemaker_client.describe_labeling_job(LabelingJobName=job_name)
['LabelingJobOutput']['FinalActiveLearningModelArn']
```

To encrypt data on the storage volume attached to the ML compute instance(s) that are used in automated labeling, include an Amazon Key Management Service (Amazon KMS) key in the `VolumeKmsKeyId` parameter. For information about Amazon KMS keys, see [What is Amazon Key Management Service?](#) in the [Amazon Key Management Service Developer Guide](#).

For an example that uses the `CreateLabelingJob` operation to create an automated data labeling job, see the `object_detection_tutorial` example in the [SageMaker Examples, Ground Truth Labeling Jobs](#) section of a SageMaker notebook instance. To learn how to create and open a notebook instance, see [Create a Notebook Instance \(p. 120\)](#). To learn how to access SageMaker example notebooks, see [Example Notebooks \(p. 131\)](#).

## Amazon EC2 Instances Required for Automated Data Labeling

The following table lists the Amazon Elastic Compute Cloud (Amazon EC2) instances that you need to run automated data labeling for training and batch inference jobs.

Automated Data Labeling Job Type	Training Instance Type	Inference Instance Type
Image classification	ml.p3.2xlarge*	ml.c5.xlarge
Object detection (bounding box)	ml.p3.2xlarge*	ml.c5.4large
Text classification	ml.c5.2xlarge	ml.m4.xlarge
Semantic segmentation	ml.p3.2xlarge*	ml.p3.2xlarge*

\* In the Asia Pacific (Mumbai) Region (ap-south-1) use ml.p2.8xlarge instead.

Automated data labeling incurs two separate charges: the per-item charge (see [pricing](#)), and the charge for the Amazon EC2 instance required to run the model (see [Amazon EC2 pricing](#)).

Ground Truth manages the instances that you use for automated data labeling jobs. It creates, configures, and terminates the instances as needed to perform your job. These instances don't appear in your Amazon EC2 instance dashboard.

## Set up an active learning workflow with your own model

You can create an active learning workflow with your own algorithm to run training and inferences in that workflow to auto-label your data. The notebook `bring_your_own_model_for_sagemaker_labeling_workflows_with_active_learning.ipynb` demonstrates this using the SageMaker built-in algorithm, [BlazingText](#). This notebook provides an Amazon CloudFormation stack that you can use to execute this workflow using Amazon Step Functions. You can find the notebook and supporting files in this [GitHub repository](#).

You can also find this notebook in the SageMaker Examples repository. See [Use Example Notebooks](#) to learn how to find an Amazon SageMaker example notebook.

## Chaining Labeling Jobs

Amazon SageMaker Ground Truth can reuse datasets from prior jobs in two ways: cloning and chaining.

*Cloning* copies the setup of a prior labeling job and allows you to make additional changes before setting it to run.

*Chaining* uses not only the setup of the prior job, but also the results. This allows you to continue an incomplete job and add labels or data objects to a completed job. Chaining is a more complex operation.

For data processing:

- Cloning uses the prior job's *input* manifest, with optional modifications, as the new job's input manifest.
- Chaining uses the prior job's *output* manifest as the new job's input manifest.

Chaining is useful when you need to:

- Continue a labeling job that was manually stopped.
- Continue a labeling job that failed mid-job, after fixing issues.
- Switch to automated data labeling after manually labeling part of a job (or the other way around).
- Add more data objects to a completed job and start the job from there.

- Add another annotation to a completed job. For example, you have a collection of phrases labeled for topic, then want to run the set again, categorizing them by the topic's implied audience.

In Amazon SageMaker Ground Truth you can configure a chained labeling job with either the console or the API.

### Key Term: Label Attribute Name

The *label attribute name* (`LabelAttributeName` in the API) is a string used as the key for the key-value pair formed with the label that a worker assigns to the data object.

The following rules apply for the label attribute name:

- It can't end with `-metadata`.
- The names `source` and `source-ref` are reserved and can't be used.
- For semantic segmentation labeling jobs, it must end with `-ref`. For all other labeling jobs, it can't end with `-ref`. If you use the console to create the job, Amazon SageMaker Ground Truth automatically appends `-ref` to all label attribute names except for semantic segmentation jobs.
- For a chained labeling job, if you're using the same label attribute name from the originating job and you configure the chained job to use auto-labeling, then if it had been in auto-labeling mode at any point, Ground Truth uses the model from the originating job.

In an output manifest, the label attribute name appears similar to the following.

```
"source-ref": "<S3 URI>",
"<label attribute name>": {
  "annotations": [
    {
      "class_id": 0,
      "width": 99,
      "top": 87,
      "height": 62,
      "left": 175
    }
  ],
  "image_size": [
    {
      "width": 344,
      "depth": 3,
      "height": 234
    }
  ],
  "<label attribute name>-metadata": {
    "job-name": "<job name>",
    "class-map": {
      "0": "<label attribute name>"
    },
    "human-annotated": "yes",
    "objects": [
      {
        "confidence": 0.09
      }
    ],
    "creation-date": "<timestamp>",
    "type": "groundtruth/object-detection"
  }
}
```

If you're creating a job in the console and don't explicitly set the label attribute name value, Ground Truth uses the job name as the label attribute name for the job.

### Start a Chained Job (Console)

Choose a stopped, failed, or completed labeling job from the list of your existing jobs. This enables the **Actions** menu.

From the **Actions** menu, choose **Chain**.

### Job Overview Panel

In the **Job overview** panel, a new **Job name** is set based on the title of the job from which you are chaining this one. You can change it.

You may also specify a label attribute name different from the labeling job name.

If you're chaining from a completed job, the label attribute name uses the name of the new job you're configuring. To change the name, select the check box.

If you're chaining from a stopped or failed job, the label attribute name uses to the name of the job from which you're chaining. It's easy to see and edit the value because the name check box is checked.

### Attribute label naming considerations

- **The default** uses the label attribute name Ground Truth has selected. All data objects without data connected to that label attribute name are labeled.
- **Using a label attribute name** not present in the manifest causes the job to process *all* the objects in the dataset.

The **input dataset location** in this case is automatically selected as the output manifest of the chained job. The input field is not available, so you cannot change it.

### Adding data objects to a labeling job

You cannot specify an alternate manifest file. Manually edit the output manifest from the previous job to add new items before starting a chained job. The Amazon S3 URI helps you locate where you are storing the manifest in your Amazon S3 bucket. Download the manifest file from there, edit it locally on your computer, and then upload the new version to replace it. Make sure you are not introducing errors during editing. We recommend you use JSON linter to check your JSON. Many popular text editors and IDEs have linter plugins available.

### Start a Chained Job (API)

The procedure is almost the same as setting up a new labeling job with `CreateLabelingJob`, except for two primary differences:

- **Manifest location:** Rather than use your original manifest from the prior job, the value for the `ManifestS3Uri` in the `DataSource` should point to the Amazon S3 URI of the *output manifest* from the prior labeling job.
- **Label attribute name:** Setting the correct `LabelAttributeName` value is important here. This is the key portion of a key-value pair where labeling data is the value. Sample use cases include:
  - **Adding new or more specific labels to a completed job** — Set a new label attribute name.
  - **Labeling the unlabeled items from a prior job** — Use the label attribute name from the prior job.

### Use a Partially Labeled Dataset

You can get some chaining benefits if you use an augmented manifest that has already been partially labeled. Check the **Label attribute name** check box and set the name so that it matches the name in your manifest.

If you're using the API, the instructions are the same as those for starting a chained job. However, be sure to upload your manifest to an Amazon S3 bucket and use it instead of using the output manifest from a prior job.

The **Label attribute name** value in the manifest has to conform to the naming considerations discussed earlier.

## Ground Truth Security and Permissions

Use the topics on this page to learn about Ground Truth security features and how to configure Amazon Identity and Access Management (IAM) permissions to allow an IAM user or role to create a labeling job. Additionally, learn how to create an *execution role*. An execution role is the role that you specify when you create a labeling job. This role is used to start your labeling job.

If you are a new user and want to get started quickly, or if you do not require granular permissions, see [Use IAM Managed Policies with Ground Truth \(p. 440\)](#).

For more information about IAM users and roles, see [Identities \(Users, Groups, and Roles\)](#) in the IAM User Guide.

To learn more about using IAM with SageMaker, see [Identity and Access Management for Amazon SageMaker \(p. 2417\)](#).

### Topics

- [CORS Permission Requirement \(p. 439\)](#)
- [Assign IAM Permissions to Use Ground Truth \(p. 440\)](#)
- [Output Data and Storage Volume Encryption \(p. 451\)](#)
- [Workforce Authentication and Restrictions \(p. 452\)](#)

## CORS Permission Requirement

Earlier in 2020, widely used browsers like Chrome and Firefox changed their default behavior for rotating images based on image metadata, referred to as [EXIF data](#). Previously, browsers would always display images in exactly the manner in which they are stored on disk, which is typically unrotated. After the change, images now rotate according to a piece of image metadata called *orientation value*. This has important implications for the entire machine learning (ML) community. For example, if applications that annotate images do not consider the EXIF orientation, they may display images in unexpected orientations, resulting in incorrect labels.

Starting with Chrome 89, Amazon can no longer automatically prevent the rotation of images because the web standards group W3C has decided that the ability to control rotation of images violates the web's Same-origin Policy. Therefore, to ensure human workers annotate your input images in a predictable orientation when you submit requests to create a labeling job, you must add a CORS header policy to the Amazon S3 buckets that contain your input images.

### Important

If you do not add a CORS configuration to the Amazon S3 buckets that contain your input data, labeling tasks for those input data objects will fail.

If you create a job through the Ground Truth console, CORS is enabled by default. If all of your input data is *not* located in the same Amazon S3 bucket as your input manifest file, you must add a CORS configuration to all Amazon S3 buckets that contain input data using the following instructions.

If you are using the `CreateLabelingJob` API to create a Ground Truth labeling job, you can add a CORS policy to an Amazon S3 bucket that contains input data in the S3 console. To set the required CORS headers on the Amazon S3 bucket that contain your input images in the Amazon S3 console, follow the directions detailed in [How do I add cross-domain resource sharing with CORS?](#). Use the following CORS configuration code for the buckets that host your images. If you use the Amazon S3 console to add the policy to your bucket, you must use the JSON format.

### Important

If you create a 3D point cloud or video frame labeling job, you must add additional rules to your CORS configuration. To learn more, see [3D Point Cloud Labeling Job Permission Requirements \(p. 262\)](#) and [Video Frame Job Permission Requirements \(p. 213\)](#) respectively.

## JSON

```
[  
  {  
    "AllowedHeaders": [],  
    "AllowedMethods": ["GET"],  
    "AllowedOrigins": ["*"],  
    "ExposeHeaders": []  
}]
```

## XML

```
<corsConfiguration>  
  <corsRule>  
    <allowedOrigin>*</allowedOrigin>  
    <allowedMethod>GET</allowedMethod>  
  </corsRule>  
</corsConfiguration>
```

## Assign IAM Permissions to Use Ground Truth

Use the topics in this section to learn how to use Amazon Identity and Access Management (IAM) managed and custom policies to manage access to Ground Truth and associated resources.

You can use the sections on this page to learn the following:

- How to create IAM policies that grant an IAM user or role permission to create a labeling job. Administrators can use IAM policies to restrict access to Amazon SageMaker and other Amazon services that are specific to Ground Truth.
- How to create a SageMaker *execution role*. An execution role is the role that you specify when you create a labeling job. The role is used to start and manage your labeling job.

The following is an overview of the topics you'll find on this page:

- If you are getting started using Ground Truth, or you do not require granular permissions for your use case, it is recommended that you use the IAM managed policies described in [Use IAM Managed Policies with Ground Truth \(p. 440\)](#).
- Learn about the permissions required to use the Ground Truth console in [Grant IAM Permission to Use the Amazon SageMaker Ground Truth Console \(p. 441\)](#). This section includes policy examples that grant an IAM entity permission to create and modify private work teams, subscribe to vendor work teams, and create custom labeling workflows.
- When you create a labeling job, you must provide an execution role. Use [Create a SageMaker Execution Role for a Ground Truth Labeling Job \(p. 445\)](#) to learn about the permissions required for this role.

## Use IAM Managed Policies with Ground Truth

SageMaker and Ground Truth provide Amazon managed policies that you can use to create a labeling job. If you are getting started using Ground Truth and you do not require granular permissions for your use case, it is recommended that you use the following policies:

- [AmazonSageMakerFullAccess](#) – Use this policy to give an IAM user or role permission to create a labeling job. This is a broad policy that grants an IAM entity permission to use SageMaker features, as well as features of necessary Amazon services through the console and API. This policy gives the IAM entity permission to create a labeling job and to create and manage workforces using Amazon Cognito. To learn more, see [AmazonSageMakerFullAccess Policy](#).
- [AmazonSageMakerGroundTruthExecution](#) – To create an *execution role*, you can attach the policy [AmazonSageMakerGroundTruthExecution](#) to an IAM role. An execution role is the role that you

specify when you create a labeling job and it is used to start your labeling job. This policy allows you to create both streaming and non-streaming labeling jobs, and to create a labeling job using any task type. Note the following limits of this managed policy.

- **Amazon S3 permissions:** This policy grants an execution role permission to access Amazon S3 buckets with the following strings in the name: GroundTruth, Groundtruth, groundtruth, SageMaker, Sagemaker, and sagemaker or a bucket with an [object tag](#) that includes SageMaker in the name (case insensitive). Make sure your input and output bucket names include these strings, or add additional permissions to your execution role to [grant it permission to access your Amazon S3 buckets](#). You must give this role permission to perform the following actions on your Amazon S3 buckets: AbortMultipartUpload, GetObject, and PutObject.
- **Custom Workflows:** When you create a [custom labeling workflow](#), this execution role is restricted to invoking Amazon Lambda functions with one of the following strings as part of the function name: GtRecipe, SageMaker, Sagemaker, sagemaker, or LabelingFunction. This applies to both your pre-annotation and post-annotation Lambda functions. If you choose to use names without those strings, you must explicitly provide `lambda:InvokeFunction` permission to the execution role used to create the labeling job.

To learn how to attach an Amazon managed policy to an IAM user or role (identity), refer to [Adding and removing IAM identity permissions](#) in the IAM User Guide.

## Grant IAM Permission to Use the Amazon SageMaker Ground Truth Console

To use the Ground Truth area of the SageMaker console, you need to grant permission to an IAM entity to access SageMaker and other Amazon services that Ground Truth interacts with. Required permissions to access other Amazon services depends on your use-case:

- Amazon S3 permissions are required for all use cases. These permissions must grant access to the Amazon S3 buckets that contain input and output data.
- Amazon Web Services Marketplace permissions are required to use a vendor workforce.
- Amazon Cognito permission are required for private work team setup.
- Amazon KMS permissions are required to view available Amazon KMS keys that can be used for output data encryption.
- IAM permissions are required to either list pre-existing execution roles, or to create a new one. Additionally, you must use add a `PassRole` permission to allow SageMaker to use the execution role chosen to start the labeling job.

The following sections list policies you may want to grant to an IAM role to use one or more functions of Ground Truth.

### Topics

- [Ground Truth Console Permissions \(p. 441\)](#)
- [Custom Labeling Workflow Permissions \(p. 444\)](#)
- [Private Workforce Permissions \(p. 445\)](#)
- [Vendor Workforce Permissions \(p. 445\)](#)

### Ground Truth Console Permissions

To grant permission to an IAM user or role to use the Ground Truth area of the SageMaker console to create a labeling job, attach the following policy to the user or role. The following policy will give an IAM role permission to create a labeling job using a [built-in task type](#) task type. If you want to create a custom labeling workflow, add the policy in [Custom Labeling Workflow Permissions \(p. 444\)](#) to the following policy. Each Statement included in the following policy is described below this code block.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "SageMakerApis",
            "Effect": "Allow",
            "Action": [
                "sagemaker:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "KmsKeysForCreateForms",
            "Effect": "Allow",
            "Action": [
                "kms:DescribeKey",
                "kms>ListAliases"
            ],
            "Resource": "*"
        },
        {
            "Sid": "AccessAwsMarketplaceSubscriptions",
            "Effect": "Allow",
            "Action": [
                "aws-marketplace:ViewSubscriptions"
            ],
            "Resource": "*"
        },
        {
            "Sid": "SecretsManager",
            "Effect": "Allow",
            "Action": [
                "secretsmanager>CreateSecret",
                "secretsmanager>DescribeSecret",
                "secretsmanager>ListSecrets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "ListAndCreateExecutionRoles",
            "Effect": "Allow",
            "Action": [
                "iam>ListRoles",
                "iam>CreateRole",
                "iam>CreatePolicy",
                "iam>AttachRolePolicy"
            ],
            "Resource": "*"
        },
        {
            "Sid": "PassRoleForExecutionRoles",
            "Effect": "Allow",
            "Action": [
                "iam>PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "sagemaker.amazonaws.com"
                }
            }
        },
        {
            "Sid": "GroundTruthConsole",
            "Effect": "Allow",
            "Action": [
                "lambda:InvokeFunction"
            ],
            "Resource": "*"
        }
    ]
}
```

```

    "Action": [
        "groundtruthlabeling:*",
        "lambda:InvokeFunction",
        "lambda>ListFunctions",
        "s3:GetObject",
        "s3:PutObject",
        "s3>ListBucket",
        "s3:GetBucketCors",
        "s3:PutBucketCors",
        "s3>ListAllMyBuckets",
        "cognito-idp:AdminAddUserToGroup",
        "cognito-idp:AdminCreateUser",
        "cognito-idp:AdminDeleteUser",
        "cognito-idp:AdminDisableUser",
        "cognito-idp:AdminEnableUser",
        "cognito-idp:AdminRemoveUserFromGroup",
        "cognito-idp>CreateGroup",
        "cognito-idp>CreateUserPool",
        "cognito-idp>CreateUserPoolClient",
        "cognito-idp>CreateUserPoolDomain",
        "cognito-idp:DescribeUserPool",
        "cognito-idp:DescribeUserPoolClient",
        "cognito-idp>ListGroups",
        "cognito-idp>ListIdentityProviders",
        "cognito-idp>ListUsers",
        "cognito-idp>ListUsersInGroup",
        "cognito-idp>ListUserPoolClients",
        "cognito-idp>ListUserPools",
        "cognito-idp:UpdateUserPool",
        "cognito-idp:UpdateUserPoolClient"
    ],
    "Resource": "*"
}
]
}

```

This policy includes the following statements. You can scope down any of these statements by adding specific resources to the Resource list for that statement.

#### **SageMakerApis**

This statement includes `sagemaker:*`, which allows the user to perform all [SageMaker API actions](#). You can reduce the scope of this policy by restricting users from performing actions that are not used to create and monitoring a labeling job.

#### **KmsKeysForCreateForms**

You only need to include this statement if you want to grant a user permission to list and select Amazon KMS keys in the Ground Truth console to use for output data encryption. The policy above grants a user permission to list and select any key in the account in Amazon KMS. To restrict the keys that a user can list and select, specify those key ARNs in Resource.

#### **SecretsManager**

This statement gives the user permission to describe, list, and create resources in Amazon Secrets Manager required to create the labeling job.

#### **ListAndCreateExecutionRoles**

This statement gives a user permission to list (`ListRoles`) and create (`CreateRole`) IAM roles in your account. It also grants the user permission to create (`CreatePolicy`) policies and attach (`AttachRolePolicy`) policies to IAM entities. These are required to list, select, and if required, create an execution role in the console.

If you have already created an execution role, and want to narrow the scope of this statement so that users can only select that role in the console, specify the ARNs of the IAM roles you want the user to have permission to view in `Resource` and remove the actions `CreateRole`, `CreatePolicy`, and `AttachRolePolicy`.

#### **AccessAwsMarketplaceSubscriptions**

These permissions are required to view and choose vendor work teams that you are already subscribed to when creating a labeling job. To give the user permission to *subscribe* to vendor work teams, add the statement in [Vendor Workforce Permissions \(p. 445\)](#) to the policy above

#### **PassRoleForExecutionRoles**

This is required to give the labeling job creator permission to preview the worker UI and verify that input data, labels, and instructions display correctly. This statement gives an IAM entity permissions to pass the IAM execution role used to create the labeling job to SageMaker to render and preview the worker UI. To narrow the scope of this policy, add the role ARN of the execution role used to create the labeling job under `Resource`.

#### **GroundTruthConsole**

- `groundtruthlabeling` – This allows a user to perform actions required to use certain features of the Ground Truth console. These include permissions to describe the labeling job status (`DescribeConsoleJob`), list all dataset objects in the input manifest file (`ListDatasetObjects`), filter the dataset if dataset sampling is selected (`RunFilterOrSampleDatasetJob`), and to generate input manifest files if automated data labeling is used (`RunGenerateManifestByCrawlingJob`). These actions are only available when using the Ground Truth console and cannot be called directly using an API.
- `lambda:InvokeFunction` and `lambda>ListFunctions` – these actions give users permission to list and invoke Lambda functions that are used to run a custom labeling workflow.
- `s3:*` – All Amazon S3 permissions included in this statement are used to view Amazon S3 buckets for [automated data setup](#) (`ListAllMyBuckets`), access input data in Amazon S3 (`ListBucket`, `GetObject`), check for and create a CORS policy in Amazon S3 if needed (`GetBucketCors` and `PutBucketCors`), and write labeling job output files to S3 (`PutObject`).
- `cognito-idp` – These permissions are used to create, view and manage and private workforce using Amazon Cognito. To learn more about these actions, refer to the [Amazon Cognito API References](#).

#### **Custom Labeling Workflow Permissions**

Add the following statement to a policy similar to the one in [Ground Truth Console Permissions \(p. 441\)](#) to give an IAM user permission to select pre-existing pre-annotation and post-annotation Lambda functions while [creating a custom labeling workflow](#).

```
{  
    "Sid": "GroundTruthConsoleCustomWorkflow",  
    "Effect": "Allow",  
    "Action": [  
        "lambda:InvokeFunction",  
        "lambda>ListFunctions"  
    ],  
    "Resource": "*"  
}
```

To learn how to give an IAM entity permission to create and test pre-annotation and post-annotation Lambda functions, see [Required Permissions To Use Lambda With Ground Truth](#).

## Private Workforce Permissions

When added to a permissions policy, the following permission grants access to create and manage a private workforce and work team using Amazon Cognito. These permissions are not required to use an [OIDC IdP workforce](#).

```
{
    "Effect": "Allow",
    "Action": [
        "cognito-idp:AdminAddUserToGroup",
        "cognito-idp:AdminCreateUser",
        "cognito-idp:AdminDeleteUser",
        "cognito-idp:AdminDisableUser",
        "cognito-idp:AdminEnableUser",
        "cognito-idp:AdminRemoveUserFromGroup",
        "cognito-idp:CreateGroup",
        "cognito-idp:CreateUserPool",
        "cognito-idp:CreateUserPoolClient",
        "cognito-idp:CreateUserPoolDomain",
        "cognito-idp:DescribeUserPool",
        "cognito-idp:DescribeUserPoolClient",
        "cognito-idp>List",
        "cognito-idp:UpdateUserPool",
        "cognito-idp:UpdateUserPoolClient"
    ],
    "Resource": "*"
}
```

To learn more about creating private workforce using Amazon Cognito, see [Create and Manage Amazon Cognito Workforce \(p. 462\)](#).

## Vendor Workforce Permissions

You can add the following statement to the policy in [Grant IAM Permission to Use the Amazon SageMaker Ground Truth Console \(p. 441\)](#) to grant an IAM entity permission to subscribe to a vendor workforce.

```
{
    "Sid": "AccessAwsMarketplaceSubscriptions",
    "Effect": "Allow",
    "Action": [
        "aws-marketplace:Subscribe",
        "aws-marketplace:Unsubscribe",
        "aws-marketplace:ViewSubscriptions"
    ],
    "Resource": "*"
}
```

## Create a SageMaker Execution Role for a Ground Truth Labeling Job

When you configure your labeling job, you need to provide an *execution role*, which is a role that SageMaker has permission to assume to start and run your labeling job.

This role must give Ground Truth permission to access the following:

- Amazon S3 to retrieve your input data and write output data to an Amazon S3 bucket. You can either grant permission for an IAM role to access an entire bucket by providing the bucket ARN, or you can grant access to the role to access specific resources in a bucket. For example, the ARN for a bucket may look similar to `arn:aws:s3:::awsexamplebucket1` and the ARN of a resource in an Amazon S3 bucket may look similar to `arn:aws:s3:::awsexamplebucket1/prefix/file-name.png`. To

apply an action to all resources in an Amazon S3 bucket, you can use the wild card: \*. For example, `arn:aws:s3:::awsexamplebucket1/prefix/*`. For more information, see [Amazon Amazon S3 Resources](#) in the Amazon Simple Storage Service Developer Guide.

- CloudWatch to log worker metrics and labeling job statuses.
- Amazon KMS for data encryption. (Optional)
- Amazon Lambda for processing input and output data when you create a custom workflow.

Additionally, if you create a [streaming labeling job](#), this role must have permission to access:

- Amazon SQS to create an interact with an SQS queue used to [manage labeling requests](#).
- Amazon SNS to subscribe to and retrieve messages from your Amazon SNS input topic and to send messages to your Amazon SNS output topic.

All of these permissions can be granted with the [AmazonSageMakerGroundTruthExecution](#) managed policy *except*:

- Data and storage volume encryption of your Amazon S3 buckets. To learn how to configure these permissions, see [Encrypt Output Data and Storage Volume with Amazon KMS \(p. 450\)](#).
- Permission to select and invoke Lambda functions that do not include `GtRecipe`, `SageMaker`, `Sagemaker`, `sagemaker`, or `LabelingFunction` in the function name.
- Amazon S3 buckets that do not include either `GroundTruth`, `Groundtruth`, `groundtruth`, `SageMaker`, `Sagemaker`, and `sagemaker` in the prefix or bucket name or an [object tag](#) that includes `SageMaker` in the name (case insensitive).

If you require more granular permissions than the ones provided in [AmazonSageMakerGroundTruthExecution](#), use the following policy examples to create an execution role that fits your specific use case.

## Topics

- [Built-In Task Types \(Non-streaming\) Execution Role Requirements \(p. 446\)](#)
- [Built-In Task Types \(Streaming\) Execution Role Requirements \(p. 447\)](#)
- [Execution Role Requirements for Custom Task Types \(p. 449\)](#)
- [Automated Data Labeling Permission Requirements \(p. 449\)](#)

## Built-In Task Types (Non-streaming) Execution Role Requirements

The following policy grants permission to create a labeling job for a [built-in task type](#). This execution policy does not include permissions for Amazon KMS data encryption or decryption. Replace each red, italicized ARN with your own Amazon S3 ARNs.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "S3ViewBuckets",  
            "Effect": "Allow",  
            "Action": [  
                "s3>ListBucket",  
                "s3:GetBucketLocation"  
            ],  
            "Resource": [  
                "arn:aws:s3:::<input-bucket-name>",  
                "arn:aws:s3:::<output-bucket-name>"  
            ]  
        }  
    ]  
}
```

```

},
{
    "Sid": "S3GetPutObjects",
    "Effect": "Allow",
    "Action": [
        "s3:AbortMultipartUpload",
        "s3:GetObject",
        "s3:PutObject"
    ],
    "Resource": [
        "arn:aws:s3:::<input-bucket-name>/*",
        "arn:aws:s3:::<output-bucket-name>/*"
    ]
},
{
    "Sid": "CloudWatch",
    "Effect": "Allow",
    "Action": [
        "cloudwatch:PutMetricData",
        "logs>CreateLogStream",
        "logs>CreateLogGroup",
        "logs>DescribeLogStreams",
        "logs:PutLogEvents"
    ],
    "Resource": "*"
}
]
}

```

### Built-In Task Types (Streaming) Execution Role Requirements

If you create a streaming labeling job, you must add a policy similar to the following to the execution role you use to create the labeling job. To narrow the scope of the policy, replace the \* in Resource with specific Amazon resources that you want to grant the IAM role permission to access and use.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:AbortMultipartUpload",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::<input-bucket-name>/*",
                "arn:aws:s3:::<output-bucket-name>/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "s3:ExistingObjectTag/SageMaker": "true"
                }
            }
        },
        {
            "Effect": "Allow",

```

```

    "Action": [
        "s3:GetBucketLocation",
        "s3>ListBucket"
    ],
    "Resource": [
        "arn:aws:s3:::<input-bucket-name>",
        "arn:aws:s3:::<output-bucket-name>"
    ]
},
{
    "Sid": "CloudWatch",
    "Effect": "Allow",
    "Action": [
        "cloudwatch:PutMetricData",
        "logs>CreateLogStream",
        "logs>CreateLogGroup",
        "logs>DescribeLogStreams",
        "logs>PutLogEvents"
    ],
    "Resource": "*"
},
{
    "Sid": "StreamingQueue",
    "Effect": "Allow",
    "Action": [
        "sns>CreateQueue",
        "sns>DeleteMessage",
        "sns>GetQueueAttributes",
        "sns>GetQueueUrl",
        "sns>ReceiveMessage",
        "sns>SendMessage",
        "sns>SendMessageBatch",
        "sns>SetQueueAttributes"
    ],
    "Resource": "arn:aws:sqs:*::*:GroundTruth"
},
{
    "Sid": "StreamingTopicSubscribe",
    "Effect": "Allow",
    "Action": "sns:Subscribe",
    "Resource": [
        "arn:aws:sqs:<aws-region>:<aws-account-number>:<input-topic-name>",
        "arn:aws:sqs:<aws-region>:<aws-account-number>:<output-topic-name>"
    ],
    "Condition": {
        "StringEquals": {
            "sns:Protocol": "sns"
        },
        "StringLike": {
            "sns:Endpoint": "arn:aws:sqs:<aws-region>:<aws-account-number>:GroundTruth"
        }
    }
},
{
    "Sid": "StreamingTopic",
    "Effect": "Allow",
    "Action": [
        "sns:Publish"
    ],
    "Resource": [
        "arn:aws:sqs:<aws-region>:<aws-account-number>:<input-topic-name>",
        "arn:aws:sqs:<aws-region>:<aws-account-number>:<output-topic-name>"
    ]
}

```

```

        "Sid": "StreamingTopicUnsubscribe",
        "Effect": "Allow",
        "Action": [
            "sns:Unsubscribe"
        ],
        "Resource": [
            "arn:aws:sqs:<aws-region>:<aws-account-number>:<input-topic-name>",
            "arn:aws:sqs:<aws-region>:<aws-account-number>:<output-topic-name>"
        ]
    }
}

```

### Execution Role Requirements for Custom Task Types

If you want to create a [custom labeling workflow](#), add the following statement to an execution role policy like the ones found in [??? \(p. 446\)](#) or [Built-In Task Types \(Streaming\) Execution Role Requirements \(p. 447\)](#).

This policy gives the execution role permission to `Invoke` your pre-annotation and post-annotation Lambda functions.

```

{
    "Sid": "LambdaFunctions",
    "Effect": "Allow",
    "Action": [
        "lambda:InvokeFunction"
    ],
    "Resource": [
        "arn:aws:lambda:<region>:<account-id>:function:<pre-annotation-lambda-name>",
        "arn:aws:lambda:<region>:<account-id>:function:<post-annotation-lambda-name>"
    ]
}

```

### Automated Data Labeling Permission Requirements

If you want to create a labeling job with [automated data labeling](#) enabled, you must 1) add one policy to the IAM policy attached to the execution role and 2) update the trust policy of the execution role.

The following statement allows the IAM execution role to be passed to SageMaker so that it can be used to run the training and inference jobs used for active learning and automated data labeling respectively. Add this statement to an execution role policy like the ones found in [??? \(p. 446\)](#) or [Built-In Task Types \(Streaming\) Execution Role Requirements \(p. 447\)](#). Replace `arn:aws:iam:<account-number>:role/<role-name>` with the execution role ARN. You can find your IAM role ARN in the IAM console under **Roles**.

```

{
    "Effect": "Allow",
    "Action": [
        "iam:PassRole"
    ],
    "Resource": "arn:aws:iam:<account-number>:role/<execution-role-name>",
    "Condition": {
        "StringEquals": {
            "iam:PassedToService": [
                "sagemaker.amazonaws.com"
            ]
        }
    }
}

```

The following statement allows SageMaker to assume the execution role to create and manage the SageMaker training and inference jobs. This policy must be added to the trust relationship of the execution role. To learn how to add or modify an IAM role trust policy, see [Modifying a role](#) in the IAM User Guide.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {"Service": "sagemaker.amazonaws.com"},
            "Action": "sts:AssumeRole"
        }
    ]
}
```

## Encrypt Output Data and Storage Volume with Amazon KMS

You can use Amazon Key Management Service (Amazon KMS) to encrypt output data from a labeling job by specifying an [Amazon KMS customer managed customer master key \(CMK\)](#) when you create the labeling job. If you use the API operation `CreateLabelingJob` to create a labeling job that uses automated data labeling, you can also use a customer managed CMK to encrypt the storage volume attached to the ML compute instances to run the training and inference jobs.

This section describes the IAM policies you must attach to your customer managed CMK to enable output data encryption and the policies you must attach to your CMK and execution role to use storage volume encryption. To learn more about these options, see [Output Data and Storage Volume Encryption \(p. 451\)](#).

### Encrypt Output Data using KMS

If you specify an Amazon KMS customer managed CMK to encrypt output data, you must add an IAM policy similar to the following to that key. This policy gives the IAM execution role that you use to create your labeling job permission to use this key to perform all of the actions listed in "Action". To learn more about these actions, see [Amazon KMS permissions](#) in the Amazon Key Management Service Developer Guide.

To use this policy, replace the IAM service-role ARN in "Principal" with the ARN of the execution role you use to create the labeling job. When you create a labeling job in the console, this is the role you specify for **IAM Role** under the **Job overview** section. When you create a labeling job using `CreateLabelingJob`, this is ARN you specify for `RoleArn`.

```
{
    "Sid": "AllowUseOfKmsKey",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/service-role/example-role"
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
    ],
    "Resource": "*"
}
```

### Encrypt Automated Data Labeling ML Compute Instance Storage Volume

If you specify a `VolumeKmsKeyId` to encrypt the storage volume attached to the ML compute instance used for automated data labeling training and inference, you must do the following:

- Attach permissions described in [Encrypt Output Data using KMS \(p. 450\)](#) to the customer managed CMK.
- Attach a policy similar to the following to the IAM execution role you use to create your labeling job. This is the IAM role you specify for `RoleArn` in `CreateLabelingJob`. To learn more about the "kms:CreateGrant" action that this policy permits, see [CreateGrant](#) in the Amazon Key Management Service API Reference.

```
{
  "Version": "2012-10-17",
  "Statement":
  [
    {
      "Effect": "Allow",
      "Action": [
        "kms:CreateGrant"
      ],
      "Resource": "*"
    }
  ]
}
```

To learn more about Ground Truth storage volume encryption, see [Use Your KMS Key to Encrypt Automated Data Labeling Storage Volume \(API Only\) \(p. 452\)](#).

## Output Data and Storage Volume Encryption

With Amazon SageMaker Ground Truth, you can label highly sensitive data, stay in control of your data, and employ security best practices. While your labeling job is running, Ground Truth encrypts data in transit and at rest. Additionally, you can use Amazon Key Management Service (Amazon KMS) with Ground Truth to do the following:

- Use an [Amazon KMS customer managed customer master key \(CMK\)](#) to encrypt your output data.
- Use Amazon KMS customer managed CMK with your automated data labeling job to encrypt the storage volume attached to the compute instance used for model training and inference.

Use the topics on this page to learn more about these Ground Truth security features.

### Use Your KMS Key to Encrypt Output Data

Optionally, you can provide an Amazon KMS customer managed CMK when you create a labeling job, which Ground Truth uses to encrypt your output data.

If you don't provide a customer managed key, Amazon SageMaker uses the default Amazon managed CMK for Amazon S3 for your role's account to encrypt your output data.

If you provide a customer managed CMK, you must add the required permissions to the key described in [Encrypt Output Data and Storage Volume with Amazon KMS \(p. 450\)](#). When you use the API operation `CreateLabelingJob`, you can specify your CMK's ID using the parameter `KmsKeyId`. See the following procedure to learn how to add a customer managed CMK when you create a labeling job using the console.

#### To add an Amazon KMS key to encrypt output data (console):

1. Complete the first 7 steps in [Create a Labeling Job \(Console\) \(p. 336\)](#).
2. In step 8, select the arrow next to **Additional configuration** to expand this section.
3. For **Encryption key**, select the Amazon KMS key that you want to use to encrypt output data.

4. Complete the rest of steps in [Create a Labeling Job \(Console\) \(p. 336\)](#) to create a labeling job.

## Use Your KMS Key to Encrypt Automated Data Labeling Storage Volume (API Only)

When you create a labeling job with automated data labeling using the `CreateLabelingJob` API operation, you have the option to encrypt the storage volume attached to the ML compute instances that run the training and inference jobs. To add encryption to your storage volume, use the parameter `VolumeKmsKeyId` to input an Amazon KMS customer managed CMK. For more information about this parameter, see [LabelingJobResourceConfig](#).

If you specify a key ID or ARN for `VolumeKmsKeyId`, your SageMaker execution role must include permissions to call `kms:CreateGrant`. To learn how to add this permission to an execution role, see [Create a SageMaker Execution Role for a Ground Truth Labeling Job \(p. 445\)](#).

### Note

If you specify an Amazon KMS customer managed CMK when you create a labeling job in the console, that key is *only* used to encrypt your output data. It is not used to encrypt the storage volume attached to the ML compute instances used for automated data labeling.

## Workforce Authentication and Restrictions

Ground Truth enables you to use your own private workforce to work on labeling jobs. A *private workforce* is an abstract concept which refers to a set of people who work for you. Each labeling job is created using a work team, composed of workers in your workforce. Ground Truth supports private workforce creation using Amazon Cognito.

A Ground Truth workforce maps to a Amazon Cognito user pool. A Ground Truth work team maps to a Amazon Cognito user group. Amazon Cognito manages the worker authentication. Amazon Cognito supports Open ID connection (OIDC) and customers can set up Amazon Cognito federation with their own identity provider (IdP).

Ground Truth only allows one workforce per account per Amazon Region. Each workforce has a dedicated Ground Truth work portal login URL.

You can also restrict workers to a Classless Inter-Domain Routing (CIDR) block/IP address range. This means annotators must be on a specific network to access the annotation site. You can add up to ten CIDR blocks for one workforce. To learn more, see [Manage Private Workforce Using the Amazon SageMaker API \(p. 478\)](#).

To learn how you can create a private workforce, see [Create a Private Workforce \(Amazon Cognito\) \(p. 462\)](#).

## Restrict Access to Workforce Types

Amazon SageMaker Ground Truth work teams fall into one of three [workforce types](#): public (with Amazon Mechanical Turk), private, and vendor. To restrict IAM user access to a specific work team using one of these types or the work team ARN, use the `sagemaker:WorkteamType` and/or the `sagemaker:WorkteamArn` condition keys. For the `sagemaker:WorkteamType` condition key, use [string condition operators](#). For the `sagemaker:WorkteamArn` condition key, use [Amazon Resource Name \(ARN\) condition operators](#). If the user attempts to create a labeling job with a restricted work team, SageMaker returns an access denied error.

The policies below demonstrate different ways to use the `sagemaker:WorkteamType` and `sagemaker:WorkteamArn` condition keys with appropriate condition operators and valid condition values.

The following example uses the `sagemaker:WorkteamType` condition key with the `StringEquals` condition operator to restrict access to a public work team. It accepts condition values in the following format: `workforcetype-crowd`, where `workforcetype` can equal `public`, `private`, or `vendor`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RestrictWorkteamType",
            "Effect": "Deny",
            "Action": "sagemaker:CreateLabelingJob",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:WorkteamType": "public-crowd"
                }
            }
        }
    ]
}
```

The following policies show how to restrict access to a public work team using the `sagemaker:WorkteamArn` condition key. The first shows how to use it with a valid IAM regex-variant of the work team ARN and the `ArnLike` condition operator. The second shows how to use it with the `ArnEquals` condition operator and the work team ARN.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RestrictWorkteamType",
            "Effect": "Deny",
            "Action": "sagemaker:CreateLabelingJob",
            "Resource": "*",
            "Condition": {
                "ArnLike": {
                    "sagemaker:WorkteamArn": "arn:aws:sagemaker:*:::workteam/public-crowd/*"
                }
            }
        }
    ]
}
```

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RestrictWorkteamType",
            "Effect": "Deny",
            "Action": "sagemaker:CreateLabelingJob",
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "sagemaker:WorkteamArn": "arn:aws:sagemaker:us-west-2:394669845002:workteam/public-crowd/default"
                }
            }
        }
    ]
}
```

## Monitor Labeling Job Status

To monitor the status of your labeling jobs, you can set up an [Amazon CloudWatch Events](#) (CloudWatch Events) rule for Amazon SageMaker Ground Truth (Ground Truth) to send an event to CloudWatch Events when a labeling job status changes to `Completed`, `Failed`, or `Stopped`.

Once you create a rule, you can add a *target* to it. CloudWatch Events uses this target to invoke another Amazon service to process the event. For example, you can create a target using a Amazon Simple Notification Service (Amazon SNS) topic to send a notification to your email when a labeling job status changes.

### Prerequisites:

To create a CloudWatch Events rule, you will need an Amazon Identity and Access Management (IAM) role with an `events.amazonaws.com` trust policy attached. The following is an example of an `events.amazonaws.com` trust policy.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "",  
            "Effect": "Allow",  
            "Principal": {  
                "Service": [  
                    "events.amazonaws.com"  
                ]  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

### Topics

- [Send Events to CloudWatch Events \(p. 454\)](#)
- [Set Up a Target to Process Events \(p. 455\)](#)
- [Labeling Job Expiration \(p. 456\)](#)
- [Declining Tasks \(p. 456\)](#)

## Send Events to CloudWatch Events

To configure a CloudWatch Events rule to get status updates, or *events*, for your Ground Truth labeling jobs, use the Amazon Command Line Interface (Amazon CLI) `put-rule` command. You can filter events that are sent to your rule by status change. For example, you can create a rule that notifies you only if a labeling job status changes to `Completed`. When using the `put-rule` command, specify the following to receive labeling job statuses:

- `\"source\":[\"aws.sagemaker\"]`
- `\"detail-type\":[\"SageMaker Ground Truth Labeling Job State Change\"]`

To configure a CloudWatch Events rule to watch for all status changes, use the following command and replace the placeholder text. For example, replace `"GTLabelingJobStateChanges"` with a unique CloudWatch Events rule name and `"arn:aws:iam::111122223333:role/MyRoleForThisRule"` with the Amazon Resource Number (ARN) of an IAM role with an `events.amazonaws.com` trust policy attached.

```
aws events put-rule --name "GTLabelingJobStateChanges"
--event-pattern "{\"source\":[\"aws.sagemaker\"], \"detail-type\":[\"SageMaker Ground
Truth Labeling Job State Change\"]}"
--role-arn "arn:aws:iam::111122223333:role/MyRoleForThisRule"
--region "region"
```

To filter by job status, use the `\"detail\":{\"LabelingJobStatus\":[\"Status\"]}}` syntax. Valid values for `Status` are `Completed`, `Failed`, and `Stopped`.

The following example creates a CloudWatch Events rule that notifies you when a labeling job in us-west-2 (Oregon) changes to `Completed`.

```
aws events put-rule --name "LabelingJobCompleted"
--event-pattern "{\"source\":[\"aws.sagemaker\"], \"detail-type\":[\"SageMaker Ground
Truth Labeling Job State Change\"], \"detail\":{\"LabelingJobStatus\":[\"Completed\"]}}"
--role-arn "arn:aws:iam::111122223333:role/MyRoleForThisRule"
--region us-west-2
```

The following example creates a CloudWatch Events rule that notifies you when a labeling job in us-east-1 (Virginia) changes to `Completed` or `Failed`.

```
aws events put-rule --name "LabelingJobCompletedOrFailed"
--event-pattern "{\"source\":[\"aws.sagemaker\"], \"detail-type\":[\"SageMaker Ground
Truth Labeling Job State Change\"], \"detail\":{\"LabelingJobStatus\":[\"Completed\",
\"Failed\"]}}"
--role-arn "arn:aws:iam::111122223333:role/MyRoleForThisRule"
--region us-east-1
```

To learn more about the `put-rule` request, see [Event Patterns in CloudWatch Events](#) in the *Amazon CloudWatch Events User Guide*.

## Set Up a Target to Process Events

After you have created a rule, events similar to the following are sent to CloudWatch Events. In this example, the labeling job `test-labeling-job`'s status changed to `Completed`.

```
{
  "version": "0",
  "id": "111e1111-11d1-111f-b111-1111b11dc11",
  "detail-type": "SageMaker Ground Truth Labeling Job State Change",
  "source": "aws.sagemaker",
  "account": "111122223333",
  "time": "2018-10-06T12:26:13Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:sagemaker:us-east-1:111122223333:labeling-job/test-labeling-job"
  ],
  "detail": {
    "LabelingJobStatus": "Completed"
  }
}
```

To process events, you need to set up a target. For example, if you want to receive an email when your labeling job status changes, use a procedure in [Setting Up Amazon SNS Notifications](#) in the *Amazon CloudWatch User Guide* to set up an Amazon SNS topic and subscribe your email to it. Once you have create a topic, you can use it to create a target.

### To add a target to your CloudWatch Events rule

1. Open the CloudWatch console: <https://console.amazonaws.cn/cloudwatch/home>

2. In the navigation pane, choose **Rules**.
3. Choose the rule that you want to add a target to.
4. Choose **Actions**, and then choose **Edit**.
5. Under **Targets**, choose **Add Target** and choose the Amazon service you want to act when a labeling job status change event is detected.
6. Configure your target. For instructions, see the topic for configuring a target in the [Amazon documentation for that service](#).
7. Choose **Configure details**.
8. For **Name**, enter a name and, optionally, provide details about the purpose of the rule in **Description**.
9. Make sure that the check box next to **State** is selected so that your rule is listed as **Enabled**.
10. Choose **Update rule**.

## Labeling Job Expiration

If your labeling job is not completed after 30 days, it will expire. If your labeling job expires, you can chain the job to create a new labeling job that will only send unlabeled data to workers. For more information, and to learn how to create a labeling job using chaining, see [Chaining Labeling Jobs \(p. 436\)](#).

## Declining Tasks

Workers are able to decline tasks.

Workers decline a task if the instructions are not clear, input data is not displaying correctly, or if they encounter some other issue with the task. If the number of workers per dataset object ([NumberOfHumanWorkersPerDataObject](#)) decline the task, the data object is marked as expired and will not be sent to additional workers.

# Create and Manage Workforces

This feature is not available in the China Regions.

A *workforce* is the group of workers that you have selected to label your dataset. You can choose either the Amazon Mechanical Turk workforce, a vendor-managed workforce, or you can create your own private workforce to label or review your dataset. Whichever workforce type you choose, Amazon SageMaker takes care of sending tasks to workers.

When you use a private workforce, you also create *work teams*, a group of workers from your workforce that are assigned to specific *jobs*— [Amazon SageMaker Ground Truth](#) labeling jobs or [Amazon Augmented AI](#) human review tasks. You can have multiple work teams and can assign one or more work teams to each job.

You can use Amazon Cognito or your own private OpenID Connect (OIDC) Identity Provider (IdP) to manage your private workforce and work teams. For more information about the permissions required to manage your workforce this way, see [Permissions Required to Use the Amazon SageMaker Ground Truth Console \(p. 2427\)](#).

### Topics

- [Using the Amazon Mechanical Turk Workforce \(p. 457\)](#)
- [Managing Vendor Workforces \(p. 460\)](#)
- [Use a Private Workforce \(p. 461\)](#)

## Using the Amazon Mechanical Turk Workforce

The Amazon Mechanical Turk (Mechanical Turk) workforce provides the most workers for your [Amazon SageMaker Ground Truth](#) labeling job and [Amazon Augmented AI](#) human review task. The Amazon Mechanical Turk workforce is a world-wide resource. Workers are available 24 hours a day, 7 days a week. You typically get the fastest turnaround for your human review tasks and labeling jobs when you use the Amazon Mechanical Turk workforce.

Any Amazon Mechanical Turk workforce billing is handled as part of your Ground Truth or Amazon Augmented AI billing. You do not need to create a separate Mechanical Turk account to use the Amazon Mechanical Turk workforce.

### Important

You should not share confidential information, personal information, or protected health information with this workforce. You should not use the Amazon Mechanical Turk workforce when you use Amazon A2I in conjunction with Amazon HIPAA-eligible services, such as Amazon Textract and Amazon Rekognition, for workloads containing protected health information.

You can choose Mechanical Turk as your workforce when you create a Ground Truth labeling job or Amazon A2I human review workflow (flow definition). You can create a labeling job and a human review workflow using the SageMaker console and API.

When you use an API operation to create a labeling job or human review workflow, you use the following ARN for the Amazon Mechanical Turk workforce for your `WorkteamArn`. Replace `region` with the Amazon Region you are using to create the labeling job or human loops. For example, if you create a labeling job in US West (Oregon), replace `region` with `us-west-2`.

- `arn:aws:sagemaker:region:394669845002:workteam/public-crowd/default`

Ground Truth and Amazon A2I *require* that your input data is free of personally identifiable information (PII) when you use Mechanical Turk. If you use the Mechanical Turk workforce and do not specify that your input data is free of PII, your Ground Truth labeling jobs and Augmented AI tasks will fail. You specify that your input data is free of PII when you create a Ground Truth labeling job and when you create a Amazon A2I human loop using a built-in integration or the `StartHumanLoop` operation.

Use the following sections to learn how to use Mechanical Turk with these services.

### Topics

- [Use Mechanical Turk with Ground Truth \(p. 457\)](#)
- [Use Mechanical Turk with Amazon A2I \(p. 459\)](#)
- [When is Mechanical Turk Not Supported? \(p. 460\)](#)

## Use Mechanical Turk with Ground Truth

You can use Mechanical Turk with Ground Truth when you create a labeling job using the console, or the `CreateLabelingJob` operation.

When you create a labeling job, we recommend you adjust the number of workers that annotate each data object based on the complexity of the job and the quality that you need. Amazon SageMaker

Ground Truth uses annotation consolidation to improve the quality of the labels. More workers can make a difference in the quality of the labels for more complex labeling jobs, but might not make a difference for simpler jobs. For more information, see [Consolidate Annotations \(p. 429\)](#). Note that annotation consolidation is not supported for Amazon A2I human review workflows.

#### To use Mechanical Turk when you create a labeling job (console):

1. Use the following to create a labeling job using the Ground Truth area of the SageMaker console: [Create a Labeling Job \(Console\) \(p. 336\)](#).
2. When you are selecting **Worker types** in the **Workers** section, select **Amazon Mechanical Turk**.
3. Specify the total amount of time workers have to complete a task using **Task timeout**.
4. Specify the total amount of time a task remains available to workers in **Task expiration**. This is how long workers have to pick up a task before it fails.
5. Select the **Price per task** using the dropdown list. This is the amount of money a worker receives for completing a single task.
6. (Optional) If applicable, select **The dataset does not contain adult content**. SageMaker may restrict the Mechanical Turk workers that can view your task if it contains adult content.
7. You must read and confirm the following statement by selecting the check box to use the Mechanical Turk workforce. If your input data contains confidential information, personal information, or protected health information, you must select another workforce.

**You understand and agree that the Mechanical Turk workforce consists of independent contractors located worldwide and that you should not share confidential information, personal information, or protected health information with this workforce.**

8. (Optional) Select the check box next to **Enable automated data labeling** if you want to enable automated data labeling. To learn more about this feature, see [Automate Data Labeling \(p. 430\)](#).
9. You can specify the **Number of workers per dataset object** under **Additional configuration**. For example, if you enter 3 in this field, each data object will be labeled by 3 workers.

When you create your labeling job by selecting **Create**, your labeling tasks are sent to Mechanical Turk workers.

#### To use Mechanical Turk when you create a labeling job (API):

1. Use the following to create a labeling job using the `CreateLabelingJob` operation: [Create a Labeling Job \(API\) \(p. 338\)](#).
2. Use the following for the `WorkteamArn`. Replace `region` with the Amazon Region you are using to create the labeling job.

```
arn:aws:sagemaker:region:394669845002:workteam/public-crowd/default
```
3. Use `TaskTimeLimitInSeconds` to specify the total amount of time workers have to complete a task.
4. Use `TaskAvailabilityLifetimeInSeconds` to specify the total amount of time a task remains available to workers. This is how long workers have to pick up a task before it fails.
5. Use `NumberOfHumanWorkersPerDataObject` to specify the number of workers per dataset object.
6. Use `PublicWorkforceTaskPrice` to set the price per task. This is the amount of money a worker receives for completing a single task.
7. Use `DataAttributes` to specify that your input data is free of confidential information, personal information, or protected health information.

Ground Truth *requires* that your input data is free of personally identifiable information (PII) if you use the Mechanical Turk workforce. If you use Mechanical Turk and do not specify that your input

data is free of PII using the `FreeOfPersonallyIdentifiableInformation` flag, your labeling job will fail.

Use the `FreeOfAdultContent` flag to declare that your input data is free of adult content. SageMaker may restrict the Mechanical Turk workers that can view your task if it contains adult content.

You can see examples of how to use this API in the following notebooks, found on GitHub: [Ground Truth Jupyter Notebook Examples](#). You can access these notebooks under the SageMaker [Example Notebooks](#) (p. 131) in a [notebook instance](#).

## Use Mechanical Turk with Amazon A2I

You can specify that you want to use Mechanical Turk with Amazon A2I when you create a human review workflow, also referred to as a *flow definition*, in the console, or with the `CreateFlowDefinition` API operation. When you use this human review workflow to configure human loops, you must specify that your input data is free of PII.

### To use Mechanical Turk when you create a human review workflow (console):

1. Use the following to create a human review workflow in the Augmented AI section of the SageMaker console: [Create a Human Review Workflow \(Console\)](#) (p. 2340).
2. When you are selecting **Worker types** in the **Workers** section, select **Amazon Mechanical Turk**.
3. Select the **Price per task** using the dropdown list. This is the amount of money a worker receives for completing a single task.
4. (Optional) You can specify the **Number of workers per dataset object** under **Additional configuration**. For example, if you enter 3 in this field, each data object will be labeled by 3 workers.
5. (Optional) Specify the total amount of time workers have to complete a task using **Task timeout**.
6. (Optional) Specify the total amount of time a task remains available to workers in **Task expiration**. This is how long workers have to pick up a task before it fails.
7. Once you have created your human review workflow, you can use it to configure a human loop by providing its Amazon Resource Name (ARN) in the parameter `FlowDefinitionArn`. You configure a human loop using one of the API operations of a built-in task type, or the Amazon A2I runtime API operation, `StartHumanLoop`. To learn more, see [Create and Start a Human Loop](#) (p. 2358).

When you configure your human loop, you must specify that your input data is free of personally identifiable information (PII) using the `FreeOfPersonallyIdentifiableInformation` content classifier in `DataAttributes`. If you use Mechanical Turk and do not specify that your input data is free of PII, your human review tasks will fail.

Use the `FreeOfAdultContent` flag to declare that your input data is free of adult content. SageMaker may restrict the Mechanical Turk workers that can view your task if it contains adult content.

### To use Mechanical Turk when you create a human review workflow (API):

1. Use the following to create a human review workflow using the `CreateFlowDefinition` operation: [Create a Human Review Workflow \(API\)](#) (p. 2341).
2. Use the following for the `WorkteamArn`. Replace `region` with the Amazon Region you are using to create the labeling job.  
  
`arn:aws:sagemaker:region:394669845002:workteam/public-crowd/default`
3. Use `TaskTimeLimitInSeconds` to specify the total amount of time workers have to complete a task.

4. Use `TaskAvailabilityLifetimeInSeconds` to specify the total amount of time a task remains available to workers. This is how long workers have to pick up a task before it fails.
5. Use `TaskCount` to specify the number of workers per dataset object. For example, if you specify 3 for this parameter, each data object will be labeled by 3 workers.
6. Use `PublicWorkforceTaskPrice` to set the price per task. This is the amount of money a worker receives for completing a single task.
7. Once you have created your human review workflow, you can use it to configure a human loop by providing its Amazon Resource Name (ARN) in the parameter `FlowDefinitionArn`. You configure a human loop using one of the API operations of a built-in task type, or the Amazon A2I runtime API operation, `StartHumanLoop`. To learn more, see [Create and Start a Human Loop \(p. 2358\)](#).

When you configure your human loop, you must specify that your input data is free of personally identifiable information (PII) using the `FreeOfPersonallyIdentifiableInformation` content classifier in `DataAttributes`. If you use Mechanical Turk and do not specify that your input data is free of PII, your human review tasks will fail.

Use the `FreeOfAdultContent` flag to declare that your input data is free of adult content. SageMaker may restrict the Mechanical Turk workers that can view your task if it contains adult content.

You can see examples of how to use this API in the following notebooks, found on GitHub: [Amazon A2I Jupyter Notebook Examples](#).

## When is Mechanical Turk Not Supported?

This workforce is not supported under the following scenarios. In each scenario, you must use a `private` or `vendor` workforce.

- This workforce is not supported for Ground Truth video frame labeling jobs and 3D point cloud labeling jobs.
- You cannot use this workforce if your input data contains personally identifiable information (PII).
- Mechanical Turk is not available in some of the Amazon special regions. If applicable, refer to the documentation for your special region for more information.

## Managing Vendor Workforces

You can use a vendor-managed workforce to label your data using Amazon SageMaker Ground Truth (Ground Truth) and Amazon Augmented AI (Amazon A2I). Vendors have extensive experience in providing data labeling services for the purpose of machine learning. Vendor workforces for these two services must be created and managed separately through the Amazon SageMaker console.

Vendors make their services available via the Amazon Marketplace. You can find details of the vendor's services on their detail page, such as the number of workers and the hours that they work. You can use these details to make estimates of how much the labeling job will cost and the amount of time that you can expect the job to take. Once you have chosen a vendor you subscribe to their services using the Amazon Marketplace.

A subscription is an agreement between you and the vendor. The agreement spells out the details of the agreement, such as price, schedule, or refund policy. You work directly with the vendor if there are any issues with your labeling job.

You can subscribe to any number of vendors to meet your data annotation needs. When you create a labeling job or human review workflow you can specify that the job be routed to a specific vendor.

### Important

Before you send sensitive data to a vendor, check the vendor's security and compliance practices on their detail page and review the end user license agreement (EULA) that is part of your subscription agreement. You are responsible for ensuring that the vendor meets your compliance requirements for personal or confidential information. Do not share protected health information with this workforce.

You must use the console to subscribe to a vendor workforce. Once you have a subscription, you can use the [ListSubscribedWorkteams](#) operation to list your subscribed vendors.

### To subscribe to a vendor workforce

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
  2. Choose the appropriate page in the SageMaker console.
    - For Ground Truth labeling jobs, choose **Labeling workforces**, choose **Vendor**, and then choose **Find data labeling services**.
    - For Amazon A2I human review workflows, choose **Human review workforces**, choose **Vendor**, and then choose **Find human review services**.
  3. The console opens the Amazon Web Services Marketplace with:
    - data labeling services category selected for Ground Truth
    - human review services category selected for Amazon A2I
- Here you see a list of the vendor services available for this service.
4. Choose a vendor. The Amazon Web Services Marketplace shows detailed information about the data labeling or human review service. Use this information to determine if the vendor meets your requirements for your task.
  5. If the vendor meets your requirements, choose **Continue to subscribe**.
  6. Review the details of the subscription. If you agree to the terms, choose **Subscribe** to complete your subscription to the service.

## Use a Private Workforce

A **private workforce** is a group of workers that *you* choose. These can be employees of your company or a group of subject matter experts from your industry. For example, if the task is to label medical images, you could create a private workforce of people knowledgeable about the images in question.

Each Amazon account has access to a single private workforce per region, and the owner has the ability to create multiple **private work teams** within that workforce. A single private work team is used to complete a labeling job or human review task, or a *job*. You can assign each work team to a separate job or use a single team for multiple jobs. A single worker can be in more than one work team.

Your private workforce can either be created and managed using [Amazon Cognito](#) or your own private OpenID Connect (OIDC) Identity Provider (IdP).

If you are a new user of [Amazon SageMaker Ground Truth](#) or [Amazon Augmented AI](#) and do not require your workers to be managed with your own IdP, it is recommended that you use Amazon Cognito to create and manage your private workforce.

After you create a workforce, in addition to creating and managing work teams, you can do the following:

- [Track worker performance](#)
- [Create and manage Amazon SNS topics](#) to notify workers when labeling tasks are available

- [Manage Private Workforce Access to Tasks Using IP Addresses](#)

**Note**

Your private workforce is shared between Ground Truth and Amazon A2I. To create and manage private work teams used by Augmented AI, use the Ground Truth section of the SageMaker console.

**Topics**

- [Create and Manage Amazon Cognito Workforce \(p. 462\)](#)
- [Create and Manage OIDC IdP Workforce \(p. 469\)](#)
- [Manage Private Workforce Using the Amazon SageMaker API \(p. 478\)](#)
- [Track Worker Performance \(p. 479\)](#)
- [Create and manage Amazon SNS topics for your work teams \(p. 481\)](#)

## Create and Manage Amazon Cognito Workforce

Create and manage your private workforce using Amazon Cognito when you want to create your workforce using the Amazon SageMaker console or you don't want the overhead of managing worker credentials and authentication. When you create a private workforce with Amazon Cognito, it provides authentication, authorization, and user management for your private workers.

**Topics**

- [Create a Private Workforce \(Amazon Cognito\) \(p. 462\)](#)
- [Manage a Private Workforce \(Amazon Cognito\) \(p. 465\)](#)

### Create a Private Workforce (Amazon Cognito)

When you use Amazon Cognito, you can create a private workforce in one of the following ways:

- Create a new workforce while you are creating your labeling job. To learn how, see [Create an Amazon Cognito Workforce When Creating a Labeling Job \(p. 463\)](#).
- Create a new workforce before you create your labeling job. To learn how, see [Create an Amazon Cognito Workforce Using the Labeling Workforces Page \(p. 463\)](#).
- Import an existing workforce after creating a user pool in the Amazon Cognito console. To learn how, see [Create a Private Workforce \(Amazon Cognito Console\) \(p. 464\)](#).

Once you create a private workforce, that workforce and all work teams and workers associated with it are available to use for all Ground Truth labeling job tasks and Amazon Augmented AI human review workflows tasks.

If you are new to Amazon SageMaker and want to test Ground Truth or Amazon A2I, we suggest that you create a private work team consisting of people from your organization using the console. Use this work team when creating labeling or human review workflows (flow definitions) to test your worker UI and job workflow.

**Topics**

- [Create a Private Workforce \(Amazon SageMaker Console\) \(p. 462\)](#)
- [Create a Private Workforce \(Amazon Cognito Console\) \(p. 464\)](#)

### Create a Private Workforce (Amazon SageMaker Console)

You can create a private workforce in the Amazon SageMaker console in one of two ways:

- When creating a labeling job in the **Labeling jobs** page of the Amazon SageMaker Ground Truth section.
- Using the **Labeling workforces** page of the Amazon SageMaker Ground Truth section. If you are creating a private workforce for an Amazon A2I human review workflow, use this method.

Both of these methods also create a default work team containing all of the members of the workforce. This private workforce is available to use for both Ground Truth and Amazon Augmented AI jobs.

When you create a private workforce using the console, SageMaker uses Amazon Cognito as an identity provider for your workforce. If you want to use your own OpenID Connect (OIDC) Identity Provider (IdP) to create and manage your private workforce, you must create a workforce using the SageMaker API operation `CreateWorkforce`. To learn more, see [Create a Private Workforce \(OIDC IdP\) \(p. 469\)](#).

### [Create an Amazon Cognito Workforce When Creating a Labeling Job](#)

If you haven't created a private workforce when you create your labeling job and you choose to use private workers, you are prompted to create a work team. This will create a private workforce using Amazon Cognito.

#### **To create a workforce while creating a labeling job (console)**

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the navigation pane, choose **Labeling jobs** and fill in all required fields. For instructions on how to start a labeling job, see [Getting started \(p. 166\)](#). Choose **Next**.
3. Choose **Private** for the workforce type.
4. In the **Workers** section, enter:
  - a. The **Team name**.
  - b. Email addresses for up to 100 workforce members. Email addresses are case sensitive. Your workers must log in using the same case used when the address was initially entered. You can add additional workforce members after the job has been created.
  - c. The name of your organization. SageMaker uses this to customize the email sent to the workers.
  - d. A contact email address for workers to report issues related to the task.

When you create the labeling job, an email is sent to each worker inviting them to join the workforce. After creating the workforce, you can add, delete, and disable workers using the SageMaker console or the Amazon Cognito console.

### [Create an Amazon Cognito Workforce Using the Labeling Workforces Page](#)

To create and manage your private workforce using Amazon Cognito, you can use the **Labeling workforces** page. When following the instructions below, you have the option to create a private workforce by entering worker emails importing a pre-existing workforce from an Amazon Cognito user pool. To import a workforce, see [Create a Private Workforce \(Amazon Cognito Console\) \(p. 464\)](#).

#### **To create a private workforce using worker emails**

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the navigation pane, choose **Labeling workforces**.
3. Choose **Private**, then choose **Create private team**.
4. Choose **Invite new workers by email**.
5. Paste or type a list of up to 50 email addresses, separated by commas, into the email addresses box.

6. Enter an organization name and contact email.
7. Optionally, choose an SNS topic to which to subscribe the team so workers are notified by email when new Ground Truth labeling jobs become available. Amazon SNS notifications are supported by Ground Truth and are not supported by Augmented AI. If you subscribe workers to receive SNS notifications, they only receive notifications about Ground Truth labeling jobs. They do not receive notifications about Augmented AI tasks.
8. Click the **Create private team** button.

After you import your private workforce, refresh the page. On the **Private workforce summary** page, you can see information about the Amazon Cognito user pool for your workforce, a list of work teams for your workforce, and a list of all of the members of your private workforce.

**Note**

If you delete all of your private work teams, you have to repeat this process to use a private workforce in that region.

### Create a Private Workforce (Amazon Cognito Console)

Amazon Cognito is used to define and manage your private workforce and your work teams. It is a service that you can use to create identities for your workers and authenticate these identities with identity providers. A private workforce corresponds to a single **Amazon Cognito user pool**. Private work teams correspond to **Amazon Cognito user groups** within that user pool.

Example identity providers supported by Amazon Cognito:

- Social sign-in providers such as Facebook and Google
- OpenID Connect (OIDC) providers
- Security Assertion Markup Language (SAML) providers such as Active Directory
- The Amazon Cognito built-in identity provider

For more information, see [What Is Amazon Cognito?](#).

To create a private workforce using Amazon Cognito, you must have an existing Amazon Cognito user pool containing at least one user group. See [Tutorial: Creating a User Pool](#) to learn how to create a user pool. See [Adding Groups to a User Pool](#) to learn how to add a user group to a pool.

Once your user pool has been created, follow the steps below to create a private workforce by importing that user pool into Amazon SageMaker.

### To create a private workforce by importing a Amazon Cognito user pool

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the navigation pane, choose **Labeling workforces**.
3. Choose **Private**.
4. Choose **Create private team**. This creates a private workforce and a work team.
5. Choose **Import workers from existing Amazon Cognito user groups**.
6. Choose a user pool that you have created. User pools require a domain and an existing user group. If you get an error that the domain is missing, set it in the **Domain name** options on the **App integration** page of the Amazon Cognito console for your group.
7. Choose an app client. We recommend using a client generated by SageMaker.
8. Choose a user group from your pool to import its members.
9. Optionally choose an Amazon Simple Notification Service (Amazon SNS) topic to which to subscribe the team so that workers are notified by email when new labeling jobs become available. Amazon

SNS notifications are supported by Ground Truth and are not supported by Augmented AI. If you subscribe workers to receive SNS notifications, they only receive notifications about Ground Truth labeling jobs. They do not receive notifications about Augmented AI tasks.

10. Choose **Create private team**.

**Important**

After you create a workforce using an Amazon Cognito user pool, it should not be deleted without first deleting all work teams associated with that pool in the SageMaker console.

After you import your private workforce, refresh the page to see the **Private workforce summary** page. On this page, you can see information about the Amazon Cognito user pool for your workforce, a list of work teams for your workforce, and a list of all of the members of your private workforce. This workforce is now available to use in both Amazon Augmented AI and Amazon SageMaker Ground Truth for human review tasks and data labeling jobs respectively.

## Manage a Private Workforce (Amazon Cognito)

After you have created a private workforce using Amazon Cognito, you can create and manage work teams using the Amazon SageMaker console and API operations.

You can do the following using either the [SageMaker console](#) or [Amazon Cognito console](#).

- Add and delete work teams.
- Add workers to your workforce and one or more work teams.
- Disable or remove workers from your workforce and one or more workteams. If you add workers to a workforce using the Amazon Cognito console, you must use the same console to remove the worker from the workforce.

You can restrict access to tasks to workers at specific IP addresses using the SageMaker API. For more information, see [Manage Private Workforce Using the Amazon SageMaker API \(p. 478\)](#).

### Topics

- [Manage a Workforce \(Amazon SageMaker Console\) \(p. 465\)](#)
- [Manage a Private Workforce \(Amazon Cognito Console\) \(p. 467\)](#)

## Manage a Workforce (Amazon SageMaker Console)

You can use the Amazon SageMaker console to create and manage the work teams and individual workers that make up a private workforce.

Use a work team to assign members of your private workforce to a labeling or human review *job*. When you create your workforce using the SageMaker console, there is a work team called **Everyone-in-private-workforce** that enables you to assign your entire workforce to a job. Because an imported Amazon Cognito user pool may contain members that you don't want to include in your work teams, a similar work team is not created for Amazon Cognito user pools.

You have two choices to create a new work team:

- You can create a work team in the SageMaker console and add members from your workforce to the team.
- You can create a user group by using the Amazon Cognito console and then create a work team by importing the user group. You can import more than one user group into each work team. You manage the members of the work team by updating the user group in the Amazon Cognito console. See [Manage a Private Workforce \(Amazon Cognito Console\) \(p. 467\)](#) for more information.

## Create a Work Team Using the SageMaker Console

You can create a new Amazon Cognito user group or import an existing user group using the SageMaker console, on the **Labeling workforces** page. For more information on creating a user group in the Amazon Cognito console, see [Manage a Private Workforce \(Amazon Cognito Console\) \(p. 467\)](#).

### To create a work team using the SageMaker console

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Choose **Labeling workforces** from the left menu.
3. Under **Private**, choose **Create private team**.
4. Under **Team details**, enter a **Team name**. The name must be unique in your account in an Amazon Region.
5. Under **Add workers**, choose a method to add workers to the team using a user group.
  - If you chose **Create a team by adding workers to a new Amazon Cognito user group**, select the workers to add to the team.
  - If you chose **Create a team by importing existing Amazon Cognito user groups**, choose the user groups that are part of the new team.
6. If you select an **SNS topic**, all workers added to the team are subscribed to the Amazon SNS topic and notified when new work items are available to the team. Select from a list of your existing Ground Truth related Amazon SNS topics or select **Create new topic** to open a topic-creation dialog.

Amazon SNS notifications are supported by Ground Truth and are not supported by Augmented AI. If you subscribe workers to receive SNS notifications, they only receive notifications about Ground Truth labeling jobs. They do not receive notifications about Augmented AI tasks.

Workers in a workteam subscribed to a topic receive notifications when a new Ground Truth labeling job for that team becomes available and when one is about to expire.

Read [Create and manage Amazon SNS topics for your work teams \(p. 481\)](#) for more information about using Amazon SNS topic.

### Subscriptions

After you have created a work team, you can see more information about the team and change or set the Amazon SNS topic to which its members are subscribed by visiting the Amazon Cognito console. If you added any team members before you subscribed the team to a topic, you need to manually subscribe those members to that topic. Read [Create and manage Amazon SNS topics for your work teams](#) for more information on creating and managing the Amazon SNS topic.

### Add or Remove Workers

A *work team* is a group of workers within your workforce to whom you can assign jobs. A worker can be added to more than one work team. Once a worker has been added to a work team, that worker can be disabled or removed.

### Add Workers to the Workforce

Adding a worker to the workforce enables you to add that worker to any work team within that workforce.

### To add workers using the private workforce summary page

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Choose **Labeling workforces** to navigate to your private workforce summary page.
3. Choose **Private**.

4. Choose **Invite new workers**.
5. Paste or type a list of email addresses, separated by commas, into the email addresses box. You can have up to 50 email addresses in this list.

## Add a Worker to a Work Team

A worker must be added to the workforce before being added to a work team. To add a worker to a work team, first navigate to the **Private workforce summary** page using the steps above.

### To add a worker to a work team from the private workforce summary page

1. In the **Private teams** section, choose the team to which you want to add the workers.
2. Choose the **Workers** tab.
3. Choose **Add workers to team** and choose the boxes next to the workers that you want to add.
4. Click **Add workers to team**.

## Disable and Remove a Worker from the Workforce

Disabling a worker stops the worker from receiving a job. This action does not remove the worker from the workforce, or from any work team with which the worker is associated. To disable or remove a worker from a work team, first navigate to the private workforce summary page using the steps above.

### To deactivate a worker using the private workforce summary page

1. In the **Workers** section, choose the worker that you would like to disable.
2. Choose **Disable**.

If desired, you can subsequently **Enable** a worker after they have been disabled.

You can remove workers from your private workforce directly in the SageMaker console if that worker was added in this console. If you added the worker (user) in the Amazon Cognito console, see [Manage a Private Workforce \(Amazon Cognito Console\) \(p. 467\)](#) to learn how to remove the worker in the Amazon Cognito console.

### To remove a worker using the private workforce summary page

1. In the **Workers** section, choose the worker that you would like to delete.
2. If the worker has not been disabled, choose **Disable**.
3. Select the worker and choose **Delete**.

## Manage a Private Workforce (Amazon Cognito Console)

A private workforce corresponds to a single **Amazon Cognito user pool**. Private work teams correspond to **Amazon Cognito user groups** within that user pool. Workers correspond to **Amazon Cognito users** within those groups.

After your workforce has been created, you can add work teams and individual workers through the Amazon Cognito console. You can also delete workers from your private workforce or remove them from individual teams in the Amazon Cognito console.

#### Important

You can't delete work teams from the Amazon Cognito console. Deleting a Amazon Cognito user group that is associated with an Amazon SageMaker work team will result in an error. To remove work teams, use the SageMaker console.

## Create Work Teams (Amazon Cognito Console)

You can create a new work team to complete a job by adding a Amazon Cognito user group to the user pool associated with your private workforce. To add a Amazon Cognito user group to an existing worker pool, see [Adding groups to a User Pool](#).

### To create a work team using an existing Amazon Cognito user group

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the navigation pane, choose **Workforces**.
3. For **Private teams**, choose **Create private team**.
4. Under **Team details**, give the team a name. The name must be unique in your account in an Amazon Region.
5. For **Add workers**, choose **Import existing Amazon Cognito user groups**, and choose one or more user groups that are part of the new team.
6. If you choose an **SNS topic**, all workers added to the team are subscribed to the Amazon Simple Notification Service (Amazon SNS) topic and notified when new work items are available to the team. Choose from a list of your existing SNS topics related to SageMaker Ground Truth or Amazon Augmented AI or choose **Create new topic** to create one.

#### Note

Amazon SNS notifications are supported by Ground Truth and are not supported by Augmented AI. If you subscribe workers to receive SNS notifications, they only receive notifications about Ground Truth labeling jobs. They do not receive notifications about Augmented AI tasks.

## Subscriptions

After you have created a work team, you can see more information about the team and change or set the SNS topic to which its members are subscribed using the Amazon Cognito console. If you added any team members before you subscribed the team to a topic, you need to manually subscribe those members to that topic. For more information, see [Create and manage Amazon SNS topics for your work teams \(p. 481\)](#).

## Add and Remove Workers (Amazon Cognito Console)

When using the Amazon Cognito console to add workers to a work team, you must add a user to the user pool associated with the workforce before adding that user to a user group. Users can be added to a user pool in various ways. For more information, see [Signing Up and Confirming User Accounts](#).

### Add a Worker to a Work Team

After a user has been added to a pool, the user can be associated with user groups inside of that pool. After a user has been added to a user group, that user becomes a worker on any work team created using that user group.

### To add a user to a user group

1. Open the Amazon Cognito console: <https://console.amazonaws.cn/cognito/>.
2. Choose **Manage User Pools**.
3. Choose the user pool associated with your SageMaker workforce.
4. Under **General Settings**, choose **Users and Groups** and do one of the following:
  - Choose **Groups**, choose the group that you want to add the user to, and choose **Add users**. Choose the users that you want to add by choosing the plus-icon to the right of the user's name.
  - Choose **Users**, choose the user that you want to add to the user group, and choose **Add to group**. From the dropdown menu, choose the group and choose **Add to group**.

## Disable and Remove a Worker From a Work Team

Disabling a worker stops the worker from receiving jobs. This action doesn't remove the worker from the workforce, or from any work team the worker is associated with. To remove a user from a work team in Amazon Cognito, you remove the user from the user group associated with that team.

### To deactivate a worker (Amazon Cognito console)

1. Open the Amazon Cognito console: <https://console.amazonaws.cn/cognito/>.
2. Choose **Manage User Pools**.
3. Choose the user pool associated with your SageMaker workforce.
4. Under **General Settings**, choose **Users and Groups**.
5. Choose the user that you want to disable.
6. Choose **Disable User**.

You can enable a disabled user by choosing **Enable User**.

### To remove a user from a user group (Amazon Cognito console)

1. Open the Amazon Cognito console: <https://console.amazonaws.cn/cognito/>.
2. Choose **Manage User Pools**.
3. Choose the user pool associated with your SageMaker workforce.
4. Under **General Settings**, choose **Users and Groups**.
5. For **User** tab, choose the X icon to the right of the group from which you want to remove the user.

## Create and Manage OIDC IdP Workforce

Create a private workforce using an OpenID Connect (OIDC) Identity Provider (IdP) when you want to manage and authenticate your workers using your own OIDC IdP. Individual worker credentials and other data will be kept private. Ground Truth and Amazon A2I will only have visibility into worker information you provide through the claims that you send to these services. To create a workforce using an OIDC IdP, your IdP must support *groups* because Ground Truth and Amazon A2I map one or more groups in your IdP to a work team. To learn more, see [Send Required and Optional Claims to Ground Truth and Amazon A2I \(p. 470\)](#).

If you are a new user of Ground Truth or Amazon A2I, you can test your worker UI and job workflow by creating a private work team and adding yourself as a worker. Use this work team when you create a labeling job or human review workflow. First, create a private OIDC IdP workforce using the instructions in [Create a Private Workforce \(OIDC IdP\) \(p. 469\)](#). Next, refer to [Manage a Private Workforce \(OIDC IdP\) \(p. 475\)](#) to learn how to create a work team.

### Topics

- [Create a Private Workforce \(OIDC IdP\) \(p. 469\)](#)
- [Manage a Private Workforce \(OIDC IdP\) \(p. 475\)](#)

## Create a Private Workforce (OIDC IdP)

Create a private workforce using an OpenID Connect (OIDC) Identity Provider (IdP) when you want to authenticate and manage workers using your own identity provider. Use this page to learn how to configure your IdP to communicate with Amazon SageMaker Ground Truth (Ground Truth) or Amazon Augmented AI (Amazon A2I) and to learn how to create a workforce using your own IdP.

To create a workforce using an OIDC IdP, your IdP must support *groups* because Ground Truth and Amazon A2I use one or more groups that you specify to create work teams. You use work teams to

specify workers for your labeling jobs and human review tasks. Because groups are not a [standard claim](#), your IdP may have a different naming convention for a group of users (workers). Therefore, you must identify one or more user groups to which a worker belongs using the custom claim `sagemaker:groups` that is sent to Ground Truth or Amazon A2I from your IdP. To learn more, see [Send Required and Optional Claims to Ground Truth and Amazon A2I \(p. 470\)](#).

You create an OIDC IdP workforce using the SageMaker API operation [CreateWorkforce](#). Once you create a private workforce, that workforce and all work teams and workers associated with it are available to use for all Ground Truth labeling job tasks and Amazon A2I human review workflows tasks. To learn more, see [Create an OIDC IdP Workforce \(p. 472\)](#).

### [Send Required and Optional Claims to Ground Truth and Amazon A2I](#)

When you use your own IdP, Ground Truth and Amazon A2I use your `Issuer`, `ClientId`, and `ClientSecret` to authenticate workers by obtaining an authentication CODE from your `AuthorizationEndpoint`.

Ground Truth and Amazon A2I will use this CODE to obtain a custom claim from either your IdP's `TokenEndpoint` or `UserInfoEndpoint`. You can either configure `TokenEndpoint` to return a JSON web token (JWT) or `UserInfoEndpoint` to return a JSON object. The JWT or JSON object must contain required and optional claims that you specify. A [claim](#) is a key-value pair that contains information about a worker or metadata about the OIDC service. The following table lists the claims that must be included, and that can optionally be included in the JWT or JSON object that your IdP returns.

**Note**

Some of the parameters in the following table can be specified using a `:` or a `-`. For example, you can specify the groups a worker belongs to using `sagemaker:groups` or `sagemaker-groups` in your claim.

Name	Required	Accepted Format and Values	Description	Example
<code>sagemaker:groups</code> or <code>sagemaker-groups</code>	Yes	<p><b>Data type:</b></p> <p>If a worker belongs to a single group, identify the group using a string.</p> <p>If a worker belongs to multiple groups, use a list of up to 10 strings.</p> <p><b>Allowable characters:</b></p> <p>Regex: <code>[\p{L}\p{M}\p{S}\p{N}\p{P}]+</code></p> <p><b>Quotas:</b></p> <p>10 groups per worker</p> <p>63 characters per group name</p>	Assigns a worker to one or more groups. Groups are used to map the worker into work teams.	<p>Example of worker that belongs to a single group: "work_team1"</p> <p>Example of a worker that belongs to more than one groups: ["work_team1", "work_team2"]</p>
<code>sagemaker:sub</code> or <code>sagemaker-sub</code>	Yes	<p><b>Data type:</b></p> <p>String</p>	This is mandatory to track a worker identity inside the	"111011101-123456789-368705"

Name	Required	Accepted Format and Values	Description	Example
			<p>Ground Truth platform for auditing and to identify tasks worked on by that worker.</p> <p>For ADFS: Customers must use the Primary Security Identifier (SID).</p>	
sagemaker:client_id or sagemaker-client_id	Yes	<p><b>Data type:</b> String</p> <p><b>Allowable characters:</b> Regex: [\w+-]+</p> <p><b>Quotes:</b> 128 characters</p>	A client ID. All tokens must be issued for this client ID.	"00b600bb-1f00-05d0-bd00-00be00fb0e0"
sagemaker:name or sagemaker-name	Yes	<p><b>Data type:</b> String</p>	The worker name to be displayed in the worker portal.	"Jane Doe"
email	No	<p><b>Data type:</b> String</p>	The worker email. Ground Truth uses this email to notify workers that they have been invited to work on labeling tasks. Ground Truth will also use this email to notify your workers when labeling tasks become available if you set up an Amazon SNS topic for a work team that this worker is on.	"example-email@domain.com"
email_verified	No	<p><b>Data type:</b> Bool</p> <p><b>Accepted Values:</b> True, False</p>	Indicates if the user email was verified or not.	True

The following is an example of the JSON object syntax your UserInfoEndpoint can return.

```
{
    "sub": "122",
    "exp": "10000",
    "sagemaker-groups": [ "group1", "group2" ]}
```

```
"sagemaker-name": "name",
"sagemaker-sub": "122",
"sagemaker-client_id": "123456"
}
```

Ground Truth or Amazon A2I compares the groups listed in `sagemaker:groups` or `sagemaker-groups` to verify that your worker belongs to the work team specified in the labeling job or human review task. After the work team has been verified, labeling or human review tasks are sent to that worker.

### Create an OIDC IdP Workforce

You can create a workforce using the SageMaker API operation `CreateWorkforce` and associated language-specific SDKs. Specify a `WorkforceName` and information about your OIDC IdP in the parameter `OidcConfig`. It is recommended that you configure your OIDC with a place-holder redirect URI, and then update the URI with the worker portal URL after you create the workforce. To learn more, see [Configure your OIDC IdP \(p. 472\)](#).

The following shows an example of the request. See [CreateWorkforce](#) to learn more about each parameter in this request.

```
CreateWorkforceRequest: {
    #required fields
    WorkforceName: "example-oidc-workforce",
    OidcConfig: {
        ClientId: "clientId",
        ClientSecret: "secret",
        Issuer: "https://example-oidc-idp.com/adfs",
        AuthorizationEndpoint: "https://example-oidc-idp.com/adfs/oauth2/authorize",
        TokenEndpoint: "https://example-oidc-idp.com/adfs/oauth2/token",
        UserInfoEndpoint: "https://example-oidc-idp.com/adfs/oauth2/userInfo",
        LogoutEndpoint: "https://example-oidc-idp.com/adfs/oauth2/logout",
        JwksUri: "https://example-oidc-idp.com/adfs/discovery/keys"
    },
    SourceIpConfig: {
        Cidrs: [ "string", "string" ]
    }
}
```

### Configure your OIDC IdP

How you configure your OIDC IdP depends on the IdP you use, and your business requirements.

When you configure your IdP, you must to specify a callback or redirect URI. After Ground Truth or Amazon A2I authenticates a worker, this URI will redirect the worker to the worker portal where the workers can access labeling or human review tasks. To create a worker portal URL, you need to create a workforce with your OIDC IdP details using the `CreateWorkforce` API operation. Specifically, you must configure your OIDC IdP with required custom `sagemaker` claims (see the next section for more details). Therefore, it is recommended that you configure your OIDC with a place-holder redirect URI, and then update the URI after you create the workforce. See [Create an OIDC IdP Workforce \(p. 472\)](#) to learn how to create a workforce using this API.

You can view your worker portal URL in the SageMaker Ground Truth console, or using the SageMaker API operation, `DescribeWorkforce`. The worker portal URL is in the `SubDomain` parameter in the response.

#### Important

Make sure you add the workforce subdomain to your OIDC IdP allow list. When you add the subdomain to your allow list, it must end with `/oauth2/idpresponse`.

#### To view your worker portal URL after creating a private workforce (Console):

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.

2. In the navigation pane, choose **Labeling workforces**.
3. Select the **Private** tab.
4. In **Private workforce summary** you will see **Labeling portal sign-in URL**. This is your worker portal URL.

**To view your worker portal URL after creating a private workforce (API):**

When you create a private workforce using [CreateWorkforce](#), you specify a `WorkforceName`. Use this name to call [DescribeWorkforce](#). The following table includes examples of requests using the Amazon CLI and Amazon SDK for Python (Boto3).

SDK for Python (Boto3)

```
response = client.describe_workforce(WorkforceName='string')
print(f'The workforce subdomain is: {response['SubDomain']}')
```

Amazon CLI

```
$ C:\> describe-workforce --workforce-name 'string'
```

[Validate Your OIC IdP Workforce Authentication Response](#)

After you have created your OIDC IdP workforce, you can use the following procedure to validate its authentication workflow using cURL. This procedure assumes you have access to a terminal, and that you have cURL installed.

**To validate your OIDC IdP authorization response:**

1. Get an authorization code using a URI configured as follows:

```
{AUTHORIZE_ENDPOINT}?client_id={CLIENT_ID}&redirect_uri={REDIRECT_URI}&scope={SCOPE}&response_type=code
```

- a. Replace `{AUTHORIZE_ENDPOINT}` with the authorize endpoint for your OIDC IdP.
- b. Replace `{CLIENT_ID}` with the Client ID from your OAuth client.
- c. Replace `{REDIRECT_URI}` with the worker portal URL. If it is not already present, you must add `/oauth2/idpreponse` to the end of the URL.
- d. If you have a custom scope, use it to replace `{SCOPE}`. If you do not have a custom scope, replace `{SCOPE}` with `openid`.

The following is an example of a URI after the modifications above are made:

```
https://example.com/authorize?
client_id=f490a907-9bf1-4471-97aa-6bfd159f81ac&redirect_uri=https%3A%2F%2F
%2Fexample.labeling.sagemaker.aws
%2Foauth2%2Fidpreponse&response_type=code&scope=openid
```

2. Copy and paste the modified URI from step 1 into your browser and press Enter on your keyboard.
3. Authenticate using your IdP.
4. Copy the authentication code query parameter in the URI. This parameter begins with `code=`. The following is an example of what the response might look like. In this example, copy `code=MCNYDB...` and everything thereafter.

```
https://example.labeling.sagemaker.aws/oauth2/idpreponse?code=MCNYDB....
```

5. Open a terminal and enter the following command after making required modifications listed below:

```
$ curl --request POST \
--url '{TOKEN ENDPOINT}' \
--header 'content-type: application/x-www-form-urlencoded' \
--data grant_type=authorization_code \
--data client_id={CLIENT ID} \
--data client_secret={CLIENT SECRET} \
--data code={CODE} \
--data 'redirect_uri={REDIRECT URI}'
```

- a. Replace `{TOKEN ENDPOINT}` with the token endpoint for your OIDC IdP.
- b. Replace `{CLIENT ID}` with the Client ID from your OAuth client.
- c. Replace `{CLIENT SECRET}` with the Client Secret from your OAuth client.
- d. Replace `{CODE}` with the authentication code query parameter you copied in step 4.
- e. Replace `{REDIRECT URI}` with the worker portal URL.

The following is an example of the cURL request after making the modifications described above:

```
$ curl --request POST \
--url 'https://example.com/token' \
--header 'content-type: application/x-www-form-urlencoded' \
--data grant_type=authorization_code \
--data 'client_id=f490a907-9bf1-4471-97aa-6bfd159f81ac' \
--data client_secret=client-secret \
--data code=MCNYDB... \
--data 'redirect_uri=https://example.labeling.sagemaker.aws/oauth2/idpreponse'
```

6. This step depends on the type of `access_token` your IdP returns, a plain text access token or a JWT access token.
  - If your IdP does not support JWT access tokens, `access_token` may be plain text (for example, a UUID). The response you see may look similar to the following. In this case, move to step 7.

```
{
  "access_token": "179c144b-fccb-4d96-a28f-eea060f39c13",
  "token_type": "Bearer",
  "expires_in": 3600,
  "refresh_token": "ef43e52e-9b4f-410c-8d4c-d5c5ee57631a",
  "scope": "openid"
}
```

- If your IdP supports JWT access tokens, step 5 should generate an access token in JWT format. For example, the response may look similar to the following:

```
{
  "access_token": "eyJh...JV_adQssw5c",
  "refresh_token": "i6mapTIAVSp2oJkgUnCACKfZxt_H5MBLiqcybBBd04",
  "refresh_token_expires_in": 6327,
  "scope": "openid",
  "id_token": "eyJ0eXAiOiJK9...-rDaQzUHl6cQQWNiDpW0l_lxXjQEvoQ"
}
```

Copy the JWT and decode it. You can use python script or a third party website to decode it. For example, you can go to the website <https://jwt.io/> and paste the JWT into the **Encoded** box to decode it.

Make sure the decoded response contains the following:

- The **Required** SageMaker claims in the table found in [Send Required and Optional Claims to Ground Truth and Amazon A2I \(p. 470\)](#). If it does not, you must reconfigure your OIDC IdP to contain these claims.
- The **Issuer** you specified when you set up the IdP workforce.

7. In a terminal and enter the following command after making required modifications listed below:

```
curl -X POST -H 'Authorization: Bearer {ACCESS TOKEN}' -d '' -k -v {USERINFO ENDPOINT}
```

- a. Replace **{USERINFO ENDPOINT}** with the user info endpoint for your OIDC IdP.
- b. Replace **{ACCESS TOKEN}** with the access token in the response you received in step 7. This is the entry for the "access\_token" parameter.

The following is an example of the cURL request after making the modifications described above:

```
curl -X POST -H 'Authorization: Bearer eyJ0eX...` -d '' -k -v https://example.com/userinfo
```

8. The response to the final step in the procedure above may look similar to the following code block.

If the access\_token returned in step 6 was plain text, you must verify that this response contains required information. In this case, the response must contain the **Required** SageMaker claims in the table found in [Send Required and Optional Claims to Ground Truth and Amazon A2I \(p. 470\)](#). For example, sagemaker-groups, sagamaker-name.

```
{
    "sub": "122",
    "exp": "10000",
    "sagemaker-groups": ["group1", "group2"]
    "sagemaker-name": "name",
    "sagemaker-sub": "122",
    "sagemaker-client_id": "123456"
}
```

## Next Steps

Once you've created a private workforce using your IdP and verified your IdP authentication response, you can create work teams using your IdP groups. To learn more, see [Manage a Private Workforce \(OIDC IdP\) \(p. 475\)](#).

You can restrict worker access to tasks to specific IP addresses, and update or delete your workforce using the SageMaker API. To learn more, see [Manage Private Workforce Using the Amazon SageMaker API \(p. 478\)](#).

## Manage a Private Workforce (OIDC IdP)

Once you've created a private workforce using your OpenID Connect (OIDC) Identity Provider (IdP), you can manage your workers using your IdP. For example, you can add, remove, and group workers directly through your IdP.

To add workers to an Amazon SageMaker Ground Truth (Ground Truth) labeling job or Amazon Augmented AI (Amazon A2I) human review task, you create work teams using 1-10 IdP groups and assign that work team to the job or task. You assign a work team to a job or task by specifying that work team when you create a labeling job (Ground Truth) or a human review workflow (Amazon A2I).

You can only assign one team to each labeling job or human review workflow. You can use the same team to create multiple labeling jobs or human review tasks. You can also create multiple work teams to work on different labeling jobs or human review tasks.

## Prerequisites

To create and manage private work teams using your OIDC IdP groups, first you must create a workforce using the SageMaker API operation [CreateWorkforce](#). To learn more, see [Create a Private Workforce \(OIDC IdP\)](#) (p. 469).

## Add work teams

You can use the SageMaker console to create a private work team using your OIDC IdP workforce on the **Labeling workforces** page under **Ground Truth**. If you are creating a Ground Truth labeling job, you can also create a private work team while creating a labeling job.

### Note

You create and manage work teams for Amazon A2I in the Ground Truth area of the SageMaker console.

You can also use the SageMaker API and associated language-specific SDKs to create a private work team.

Use the following procedures to learn how to create a private work team using the SageMaker console and API.

### To create a private work team on the Labeling workforces page (console)

1. Go to the Ground Truth area of the SageMaker console: <https://console.amazonaws.cn/sagemaker/groundtruth>.
2. Select **Labeling workforces**.
3. Select **Private**.
4. In the **Private teams** section, select **Create private team**.
5. In the **Team details** section, enter a **Team name**.
6. In the **Add workers** section, enter the name of a single user group. All workers associated with this group in your IdP are added to this work team.
7. To add more than one user group, select **Add new user group** and enter the names of the user groups you want to add to this work team. Enter one user group per line.
8. (Optional) For Ground Truth labeling jobs, if you provide an email for workers in your JWT, Ground Truth notifies workers when a new labeling task is available if you select an SNS topic.
9. Select **Create private team**.

### To create a private work team while creating a Ground Truth labeling job (console)

1. Go to the Ground Truth area of the SageMaker console: <https://console.amazonaws.cn/sagemaker/groundtruth>.
2. Select **Labeling jobs**.
3. Use the instructions in [Create a Labeling Job \(Console\)](#) (p. 336) to create a labeling job. Stop when you get to the **Workers** section on the second page.

4. Select **Private** for your worker type.
5. Enter a **Team name**.
6. In the **Add workers** section, enter the name of a single user group under **User groups**. All workers associated with this group in your IdP are added to this work team.

**Important**

The group names you specify for **User groups** must match the group names specified in your OIDC IdP.

7. To add more than one user group, select **Add new user group** and enter the names of the user groups you want to add to this work team. Enter one user group per line.
8. Complete all remaining steps to create your labeling job.

The private team that you create is used for this labeling job, and is listed in the **Labeling workforces** section of the SageMaker console.

#### To create a private work team using the SageMaker API

You can create a private work team using the SageMaker API operation [CreateWorkteam](#).

When you use this operation, list all user groups that you want included in the work team in the `OidcMemberDefinition` parameter `Groups`.

**Important**

The group names you specify for `Groups` must match the group names specified in your OIDC IdP.

For example, if your user group names are `group1`, `group2`, and `group3` in your OIDC IdP, configure `OidcMemberDefinition` as follows:

```
"OidcMemberDefinition": {  
    "Groups": ["group1", "group2", "group3"]  
}
```

Additionally, you must give the work team a name using the `WorkteamName` parameter.

#### Add or remove IdP groups from work teams

After you've created a work team, you can use the SageMaker API to manage that work team. Use the [UpdateWorkteam](#) operation to update the IdP user groups included in that work team.

- Use the `WorkteamName` parameter to identify the work team that you want to update.
- When you use this operation, list all user groups that you want included in the work team in the `OidcMemberDefinition` parameter `Groups`. If a user group is associated with a work team and you do not include it in this list, that user group is no longer associated with this work team.

#### Delete a work team

You can delete a work team using the SageMaker console and SageMaker API.

#### To delete a private work team in the SageMaker console

1. Go to the Ground Truth area of the SageMaker console: <https://console.amazonaws.cn/sagemaker/groundtruth>.
2. Select **Labeling workforces**.
3. Select **Private**.

4. In the **Private teams** section, select the work team that you want to delete.
5. Select **Delete**.

#### To delete a private work team (API)

You can delete a private work team using the SageMaker API operation [DeleteWorkteam](#).

#### Manage Individual Workers

When you create a workforce using your own OIDC IdP, you cannot use Ground Truth or Amazon A2I to manage individual workers.

- To add a worker to a work team, add that worker to a group associated with that work team.
- To remove a worker from a work team, remove that worker from all user groups associated with that work team.

#### Update, Delete, and Describe Your Workforce

You can update, delete, and describe your OIDC IdP workforce using the SageMaker API. The following is a list of API operations that you can use to manage your workforce. For additional details, including how you can locate your workforce name, see [Manage Private Workforce Using the Amazon SageMaker API \(p. 478\)](#).

- [UpdateWorkforce](#) – You may want to update a workforce created using your own OIDC IdP to specify a different authorization endpoint, token endpoint, or issuer. You can update any parameter found in [OidcConfig](#) using this operation.

You can only update your OIDC IdP configuration when there are no work teams associated with your workforce. To learn how to delete work teams, see [Delete a work team \(p. 477\)](#).

- [DeleteWorkforce](#) – Use this operation to delete your private workforce. If you have any work teams associated with your workforce, you must delete those work teams before you delete your work force. For more information, see [Delete a work team \(p. 477\)](#).
- [DescribeWorkforce](#) – Use this operation to list private workforce information, including workforce name, Amazon Resource Name (ARN), and, if applicable, allowed IP address ranges (CIDRs).

## Manage Private Workforce Using the Amazon SageMaker API

You can use Amazon SageMaker API operations to manage, update, and delete your private workforce. For each API operation linked on this page, you can find a list of supported language-specific SDKs and their documentation in the **See Also** section of the API documentation.

#### Find Your Workforce Name

Some of the SageMaker workforce-related API operations require your workforce name as input. You can see your Amazon Cognito or OIDC IdP private and vendor workforce names in an Amazon Region using the [ListWorkforces](#) API operation in that Amazon Region.

If you created your workforce using your own OIDC IdP, you can find your workforce name in the Ground Truth area of the SageMaker console.

#### To find your workforce name in the SageMaker console

1. Go to the Ground Truth area of the SageMaker console: <https://console.amazonaws.cn/sagemaker/groundtruth>.

2. Select **Labeling workforces**.
3. Select **Private**.
4. In the **Private workforce summary** section, locate your workforce ARN. Your workforce name is located at the end of this ARN. For example, if the ARN is `arn:aws:sagemaker:us-east-2:111122223333:workforce/example-workforce`, the workforce name is `example-workforce`.

## Restrict Worker Access to Tasks to Allowable IP Addresses

By default, a workforce isn't restricted to specific IP addresses. You can use the [UpdateWorkforce](#) operation to require that workers use a specific range of IP addresses ([CIDRs](#)) to access tasks. If you specify one or more CIDRs, workers who attempt to access tasks using any IP address outside the specified ranges are denied and get a `Not Found` error message on the worker portal. You can specify up to 10 CIDR values using [UpdateWorkforce](#).

After you have restricted your workforce to one or more CIDRs, the output of [UpdateWorkforce](#) lists all allowable CIDRs. You can also use the [DescribeWorkforce](#) operation to view all allowable CIDRs for a workforce.

## Update OIDC Identity Provider Workforce Configuration

You may want to update a workforce created using your own OIDC IdP to specify a different authorization endpoint, token endpoint, or issuer. You can update any parameter found in [OidcConfig](#) using the [UpdateWorkforce](#) operation.

**Important**

You can only update your OIDC IdP configuration when there are no work teams associated with your workforce. You can delete a private work team using the [DeleteWorkteam](#) operation.

## Delete a Private Workforce

You can only have one private workforce in each Amazon Region. You may want to delete your private workforce in an Amazon Region when:

- You want to create a workforce using a new Amazon Cognito user pool.
- You have already created a private workforce using Amazon Cognito and you want to create a workforce using your own OpenID Connect (OIDC) Identity Provider (IdP).

To delete a private workforce, use the [DeleteWorkforce](#) API operation. If you have any work teams associated with your workforce, you must delete those work teams before you delete your workforce. You can delete a private work team using the [DeleteWorkteam](#) operation.

## Track Worker Performance

Amazon SageMaker Ground Truth logs worker events to Amazon CloudWatch, such as when a worker starts or submits a task. Use Amazon CloudWatch metrics to measure and track throughput across a team or for individual workers.

**Important**

Worker event tracking is not available for Amazon Augmented AI human review workflows.

## Enable Tracking

During the set-up process for a new work team, the permissions for Amazon CloudWatch logging of worker events are created. Since this feature was added in August 2019, work teams created prior to that may not have the correct permissions. If all of your work teams were created before August 2019, create

a new work team. It does not need any members and may be deleted after creation, but by creating it, you establish the permissions and apply them to all of your work teams, regardless of when they were created.

## Examine Logs

After tracking is enabled, the activity of your workers is logged. Open the Amazon CloudWatch console and choose **Logs** in the navigation pane. You should see a log group named **/aws/sagemaker/groundtruth/WorkerActivity**.

Each completed task is represented by a log entry, which contains information about the worker, their team, the job, when the task was accepted, and when it was submitted.

### Example Log entry

```
{  
    "worker_id": "cd449a289e129409",  
    "cognito_user_pool_id": "us-east-2_IpicJXXXX",  
    "cognito_sub_id": "d6947aeb-0650-447a-ab5d-894db61017fd",  
    "task_accepted_time": "Wed Aug 14 16:00:59 UTC 2019",  
    "task_submitted_time": "Wed Aug 14 16:01:04 UTC 2019",  
    "task_returned_time": "",  
    "task_declined_time": "",  
    "workteam_arn": "arn:aws:sagemaker:us-east-2:#####:workteam/private-crowd/Sample-labeling-team",  
    "labeling_job_arn": "arn:aws:sagemaker:us-east-2:#####:labeling-job/metrics-demo",  
    "work_requester_account_id": "#####",  
    "job_reference_code": "#####",  
    "job_type": "Private",  
    "event_type": "TasksSubmitted",  
    "event_timestamp": "1565798464"  
}
```

A useful data point in each event is the `cognito_sub_id`. You can match that to an individual worker.

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Under the **Ground Truth** section, choose **Workforces**.
3. Choose **Private**.
4. Choose the name of a team in the **Private teams** section.
5. In the **Team summary** section, choose the user group identified under **Amazon Cognito user group**. That will take you to the group in the Amazon Cognito console.
6. The **Group** page lists the users in the group. Choose any user's link in the **Username** column to see more information about the user, including a unique `sub` ID.

To get information about all of the team's members, use the `ListUsers` action ([examples](#)) in the Amazon Cognito API.

## Use Log Metrics

If you don't want to write your own scripts to process and visualize the raw log information, Amazon CloudWatch metrics provide insights into worker activity for you.

### To view metrics

1. Open the CloudWatch console at <https://console.amazonaws.cn/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.

3. Choose the AWS/SageMaker/Workteam name space, then explore the [available metrics \(p. 2523\)](#). For example, selecting the **Workteam** and **Workforce** metrics lets you calculate the average time per submitted task for a specific labeling job.

For more information, see [Using Amazon CloudWatch Metrics](#).

## Create and manage Amazon SNS topics for your work teams

Use the procedures in this topic when you want to:

- Create a topic to which you want an existing work team to subscribe.
- Create a topic before you've created a work team.
- Create or modify the work team with an API call, and specify a topic Amazon Resource Name (ARN).

If you create a work team using the console, the console provides an option to create a new topic for the team so that you don't have to perform these steps.

### Important

The Amazon SNS feature is not supported by Amazon A2I. If you subscribe your work team to an Amazon SNS topic, workers will only receive notifications about Ground Truth labeling jobs. Workers will not receive notifications about new Amazon A2I human review tasks.

### Create the Amazon SNS topic

The steps for creating Amazon SNS topics for work team notifications are similar to the steps in [Getting Started](#) in the *Amazon SNS Developer Guide*, with one significant addition—you must add an access policy so that Amazon SageMaker can publish messages to the topic on your behalf.

#### To add the policy when you create the topic

1. Open the Amazon SNS console at <https://console.amazonaws.cn/sns/v3/home>.
2. In **Create topic**, enter the name of your topic and then choose **Next steps**.
3. In **Access policy**, choose **Advanced**.
4. In the **JSON editor**, find the **Resource** property, which displays the topic's ARN.
5. Copy the **Resource** ARN value.
6. Before the final closing brace (]), add the following policy.

```
, {  
    "Sid": "AwsSagemaker_SnsAccessPolicy",  
    "Effect": "Allow",  
    "Principal": {  
        "Service": "sagemaker.amazonaws.com"  
    },  
    "Action": "sns:Publish",  
    "Resource": "ARN of the topic you copied in the previous step"  
}
```

7. Create the topic.

After you create the topic, it appears in your **Topics** summary screen. For more information about creating topics, see [Creating a Topic](#) in the *Amazon SNS Developer Guide*.

### Manage worker subscriptions

If you subscribe a work team to a topic after you've already created the work team, the individual work team members who were added to the team when the work team was created are not automatically subscribed to the topic. For information about subscribing workers' email addresses to the topic, see [Subscribing an Endpoint to an Amazon SNS Topic](#) in the *Amazon SNS Developer Guide*.

The only situation in which workers are automatically subscribed to your topic is when you create or import an Amazon Cognito user group at the time that you create a work team *and* you set up the topic subscription when you create that work team. For more information about creating and managing your workteams with Amazon Cognito, see [Create Work Teams \(Amazon Cognito Console\) \(p. 468\)](#).

## Crowd HTML Elements Reference

This feature is not available in the China Regions.

Crowd HTML Elements are web components, a web standard that abstracts HTML markup, CSS, and JavaScript functionality into an HTML tag or set of tags. Amazon SageMaker provides customers with the ability to design their own custom task templates in HTML.

As a starting point, you can use a template built using Crowd HTML Elements from one of the following GitHub repositories:

- [Example task UIs for Amazon SageMaker Ground Truth](#)
- [Over 60 example task UIs for Amazon Augmented AI \(A2I\)](#)

These repositories include templates designed for audio, image, text, video, and other types of data labeling and annotation tasks.

For more information about how to implement custom templates in Amazon SageMaker Ground Truth, see [Creating Custom Labeling Workflows \(p. 300\)](#). To learn more about custom templates in Amazon Augmented AI, see [Create Custom Worker Task Templates \(p. 2368\)](#).

## SageMaker Crowd HTML Elements

Following is a list of Crowd HTML Elements that make building a custom template easier and provide a familiar UI for workers. These elements are supported in Ground Truth, Augmented AI, and Mechanical Turk.

### Topics

- [crowd-alert \(p. 483\)](#)
- [crowd-badge \(p. 485\)](#)
- [crowd-button \(p. 486\)](#)
- [crowd-bounding-box \(p. 488\)](#)
- [crowd-card \(p. 492\)](#)
- [crowd-checkbox \(p. 494\)](#)
- [crowd-classifier \(p. 496\)](#)
- [crowd-classifier-multi-select \(p. 498\)](#)
- [crowd-entity-annotation \(p. 500\)](#)
- [crowd-fab \(p. 503\)](#)

- [crowd-form \(p. 504\)](#)
- [crowd-icon-button \(p. 505\)](#)
- [crowd-image-classifier \(p. 507\)](#)
- [crowd-image-classifier-multi-select \(p. 510\)](#)
- [crowd-input \(p. 512\)](#)
- [crowd-instance-segmentation \(p. 515\)](#)
- [crowd-instructions \(p. 518\)](#)
- [crowd-keypoint \(p. 520\)](#)
- [crowd-line \(p. 524\)](#)
- [crowd-modal \(p. 527\)](#)
- [crowd-polygon \(p. 528\)](#)
- [crowd-polyline \(p. 533\)](#)
- [crowd-radio-button \(p. 537\)](#)
- [crowd-radio-group \(p. 539\)](#)
- [crowd-semantic-segmentation \(p. 541\)](#)
- [crowd-slider \(p. 544\)](#)
- [crowd-tab \(p. 546\)](#)
- [crowd-tabs \(p. 547\)](#)
- [crowd-text-area \(p. 549\)](#)
- [crowd-toast \(p. 551\)](#)
- [crowd-toggle-button \(p. 552\)](#)

## crowd-alert

A message that alerts the worker to a current situation.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a Liquid template that uses the `<crowd-alert>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <div id="errorBox"></div>

  <crowd-keypoint
    src="{{ task.input.taskObject | grant_read_access }}"
    labels="['Item A', 'Item B', 'Item C']"
    header="Please locate the centers of each item."
    name="annotatedResult">
    <short-instructions>
      Describe your task briefly here and give examples
    </short-instructions>
    <full-instructions>
      Give additional instructions and good/bad examples here
    </full-instructions>
  </crowd-keypoint>
</crowd-form>

<script>
```

```

var num_obj = 1;

document.querySelector('crowd-form').onsubmit = function(e) {
    const keypoints = document.querySelector('crowd-keypoint').value.keypoints || document.querySelector('crowd-keypoint')._submittableValue.keypoints;
    const labels = keypoints.map(function(p) {
        return p.label;
    });

    // 1. Make sure total number of keypoints is correct.
    var original_num_labels = document.getElementsByTagName("crowd-keypoint")[0].getAttribute("labels");

    original_num_labels = original_num_labels.substring(2, original_num_labels.length - 2).split("\",\"");
    var goalNumKeypoints = num_obj*original_num_labels.length;
    if (keypoints.length != goalNumKeypoints) {
        e.preventDefault();
        errorBox.innerHTML = '<crowd-alert type="error" dismissible>You must add all keypoint annotations and use each label only once.</crowd-alert>';
        errorBox.scrollIntoView();
        return;
    }

    // 2. Make sure all labels are unique.
    labelCounts = {};
    for (var i = 0; i < labels.length; i++) {
        if (!labelCounts[labels[i]]) {
            labelCounts[labels[i]] = 0;
        }
        labelCounts[labels[i]]++;
    }
    const goalNumSingleLabel = num_obj;

    const numLabels = Object.keys(labelCounts).length;

    Object.entries(labelCounts).forEach(entry => {
        if (entry[1] != goalNumSingleLabel) {
            e.preventDefault();
            errorBox.innerHTML = '<crowd-alert type="error" dismissible>You must use each label only once.</crowd-alert>';
            errorBox.scrollIntoView();
        }
    });
};

</script>

```

## Attributes

The following attributes are supported by this element.

### **dismissible**

A Boolean switch that, if present, allows the message to be closed by the worker.

### **type**

A string that specifies the type of message to be displayed. The possible values are "info" (the default), "success", "error", and "warning".

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** none

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-badge

An icon that floats over the top right corner of another element to which it is attached.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a template that uses the `<crowd-badge>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-image-classifier
    name="crowd-image-classifier"
    src="https://unsplash.com/photos/NLUkAA-nDdE"
    header="Choose the correct category for this image."
    categories="['Person', 'Umbrella', 'Chair', 'Dolphin']"
  >
    <full-instructions header="Classification Instructions">
      <p>Read the task carefully and inspect the image.</p>
      <p>Choose the appropriate label that best suits the image.</p>
    </full-instructions>

    <short-instructions id="short-instructions">
      <p>Read the task carefully and inspect the image.</p>
      <p>Choose the appropriate label that best suits the image.</p>
      <crowd-badge icon="star" for="short-instructions"/>
    </short-instructions>
  </crowd-image-classifier>
</crowd-form>
```

## Attributes

The following attributes are supported by this element.

### for

A string that specifies the ID of the element to which the badge is attached.

### icon

A string that specifies the icon to be displayed in the badge. The string must be either the name of an icon from the open-source [iron-icons](#) set, which is pre-loaded, or the URL to a custom icon.

This attribute overrides the `label` attribute.

The following is an example of the syntax that you can use to add an iron-icon to a <crowd-badge> HTML element. Replace *icon-name* with the name of the icon you'd like to use from this [Icons set](#).

```
<crowd-badge icon="icon-name" for="short-instructions"/>
```

## label

The text to display in the badge. Three characters or less is recommended because text that is too large will overflow the badge area. An icon can be displayed instead of text by setting the *icon* attribute.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** none

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-button

A styled button that represents some action.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a template that uses the <crowd-button> element. Copy the following code and save it in a file with the extension .html. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-image-classifier
    name="crowd-image-classifier"
    src="https://unsplash.com/photos/NLUkAA-nDdE"
    header="Please select the correct category for this image"
    categories="['Person', 'Umbrella', 'Chair', 'Dolphin']"
  >
    <full-instructions header="Classification Instructions">
      <p>Read the task carefully and inspect the image.</p>
      <p>Choose the appropriate label that best suits the image.</p>
    </full-instructions>
    <short-instructions>
      <p>Read the task carefully and inspect the image.</p>
      <p>Choose the appropriate label that best suits the image.</p>
      <crowd-button>
        <iron-icon icon="question-answer"/>
      </crowd-button>
    </short-instructions>
  </crowd-image-classifier>
</crowd-form>
```

## Attributes

The following attributes are supported by this element.

### disabled

A Boolean switch that, if present, displays the button as disabled and prevents clicks.

### form-action

A switch that either submits its parent [crowd-form \(p. 504\)](#) element, if set to "submit", or resets its parent <crowd-form> element, if set to "reset".

### href

The URL to an online resource. Use this property if you need a link styled as a button.

### icon

A string that specifies the icon to be displayed next to the button's text. The string must be the name of an icon from the open-source [iron-icons](#) set, which is pre-loaded. For example, to insert the [search](#) iron-icon, use the following:

```
<crowd-button>
  <iron-icon icon="search"/>
</crowd-button>
```

The icon is positioned to either the left or the right of the text, as specified by the *icon-align* attribute.

To use a custom icon see [icon-url](#).

### icon-align

The left or right position of the icon relative to the button's text. The default is "left".

### icon-url

A URL to a custom image for the icon. A custom image can be used in place of a standard icon that is specified by the *icon* attribute.

### loading

A Boolean switch that, if present, displays the button as being in a loading state. This attribute has precedence over the *disabled* attribute if both attributes are present.

### target

When you use the *href* attribute to make the button act as a hyperlink to a specific URL, the *target* attribute optionally targets a frame or window where the linked URL should load.

### variant

The general style of the button. Use "primary" for primary buttons, "normal" for secondary buttons, "link" for tertiary buttons, or "icon" to display only the icon without text.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** none

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-bounding-box

A widget for drawing rectangles on an image and assigning a label to the portion of the image that is enclosed in each rectangle.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a Liquid template that uses the `<crowd-bounding-box>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template. For more examples, see this [GitHub repository](#).

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-bounding-box
    name="annotatedResult"
    src="{{ task.input.taskObject | grant_read_access }}"
    header="Draw bounding boxes around all the cats and dogs in this image"
    labels="['Cat', 'Dog']"
  >
    <full-instructions header="Bounding Box Instructions" >
      <p>Use the bounding box tool to draw boxes around the requested target of interest:</p>
      <ol>
        <li>Draw a rectangle using your mouse over each instance of the target.</li>
        <li>Make sure the box does not cut into the target, leave a 2 - 3 pixel margin</li>
        <li>
          When targets are overlapping, draw a box around each object,
          include all contiguous parts of the target in the box.
          Do not include parts that are completely overlapped by another object.
        </li>
        <li>
          Do not include parts of the target that cannot be seen,
          even though you think you can interpolate the whole shape of the target.
        </li>
        <li>Avoid shadows, they're not considered as a part of the target.</li>
        <li>If the target goes off the screen, label up to the edge of the image.</li>
      </ol>
    </full-instructions>

    <short-instructions>
      Draw boxes around the requested target of interest.
    </short-instructions>
  </crowd-bounding-box>
</crowd-form>
```

## Attributes

The following attributes are supported by this element.

## header

The text to display above the image. This is typically a question or simple instruction for the worker.

## initial-value

An array of JSON objects, each of which sets a bounding box when the component is loaded. Each JSON object in the array contains the following properties. Bounding boxes set via the `initial-value` property can be adjusted and whether or not a worker answer was adjusted is tracked via an `initialValueModified` boolean in the worker answer output.

- **height** – The height of the box in pixels.
- **label** – The text assigned to the box as part of the labeling task. This text must match one of the labels defined in the `labels` attribute of the `<crowd-bounding-box>` element.
- **left** – Distance of the top-left corner of the box from the left side of the image, measured in pixels.
- **top** – Distance of the top-left corner of the box from the top of the image, measured in pixels.
- **width** – The width of the box in pixels.

You can extract the bounding box initial value from a manifest file of a previous job in a custom template using the Liquid templating language:

```
initial-value=[  
    {% for box in task.input.manifestLine.label-attribute-name-from-prior-job.annotations  
    %}  
        {% capture class_id %}{{ box.class_id }}{% endcapture %}  
        {% assign label = task.input.manifestLine.label-attribute-name-from-prior-job-  
        metadata.class-map[class_id] %}  
        {  
            label: {{label | to_json}},  
            left: {{box.left}},  
            top: {{box.top}},  
            width: {{box.width}},  
            height: {{box.height}},  
        },  
    {% endfor %}  
]
```

## labels

A JSON formatted array of strings, each of which is a label that a worker can assign to the image portion enclosed by a rectangle. **Limit:** 10 labels.

## name

The name of this widget. It's used as a key for the widget's input in the form output.

## src

The URL of the image on which to draw bounding boxes.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** [full-instructions \(p. 490\)](#), [short-instructions \(p. 490\)](#)

## Regions

The following regions are required by this element.

### full-instructions

General instructions about how to draw bounding boxes.

### short-instructions

Important task-specific instructions that are displayed in a prominent place.

## Output

The following output is supported by this element.

### boundingBoxes

An array of JSON objects, each of which specifies a bounding box that has been created by the worker. Each JSON object in the array contains the following properties.

- **height** – The height of the box in pixels.
- **label** – The text assigned to the box as part of the labeling task. This text must match one of the labels defined in the *labels* attribute of the <crowd-bounding-box> element.
- **left** – Distance of the top-left corner of the box from the left side of the image, measured in pixels.
- **top** – Distance of the top-left corner of the box from the top of the image, measured in pixels.
- **width** – The width of the box in pixels.

### inputImageProperties

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

## Example : Sample Element Outputs

The following are samples of outputs from common use scenarios for this element.

### Single Label, Single Box / Multiple Label, Single Box

```
[  
  {  
    "annotatedResult": {  
      "boundingBoxes": [  
        {  
          "height": 401,  
          "label": "Dog",  
          "left": 243,  
          "top": 117,  
          "width": 187  
        }  
      ],  
      "inputImageProperties": {  
        "height": 533,  
        "width": 600  
      }  
    }  
  }  
]
```

```
        "width": 800
    }
}
]
```

### Single Label, Multiple Box

```
[
{
  "annotatedResult": {
    "boundingBoxes": [
      {
        "height": 401,
        "label": "Dog",
        "left": 243,
        "top": 117,
        "width": 187
      },
      {
        "height": 283,
        "label": "Dog",
        "left": 684,
        "top": 120,
        "width": 116
      }
    ],
    "inputImageProperties": {
      "height": 533,
      "width": 800
    }
  }
]
```

### Multiple Label, Multiple Box

```
[
{
  "annotatedResult": {
    "boundingBoxes": [
      {
        "height": 395,
        "label": "Dog",
        "left": 241,
        "top": 125,
        "width": 158
      },
      {
        "height": 298,
        "label": "Cat",
        "left": 699,
        "top": 116,
        "width": 101
      }
    ],
    "inputImageProperties": {
      "height": 533,
      "width": 800
    }
  }
]
```

You could have many labels available, but only the ones that are used appear in the output.

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-card

A box with an elevated appearance for displaying information.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a template designed for sentiment analysis tasks that uses the `<crowd-card>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<style>
  h3 {
    margin-top: 0;
  }

  crowd-card {
    width: 100%;
  }

  .card {
    margin: 10px;
  }

  .left {
    width: 70%;
    margin-right: 10px;
    display: inline-block;
    height: 200px;
  }

  .right {
    width: 20%;
    height: 200px;
    display: inline-block;
  }
</style>

<crowd-form>
  <short-instructions>
    Your short instructions here.
  </short-instructions>

  <full-instructions>
    Your full instructions here.
  </full-instructions>

  <div class="left">
    <h3>What sentiment does this text convey?</h3>
    <crowd-card>
      <div class="card">
```

```
Nothing is great.  
</div>  
</crowd-card>  
</div>  
  
<div class="right">  
<h3>Select an option</h3>  
  
<select name="sentiment1" style="font-size: large" required>  
<option value="">(Please select)</option>  
<option>Negative</option>  
<option>Neutral</option>  
<option>Positive</option>  
<option>Text is empty</option>  
</select>  
</div>  
  
<div class="left">  
<h3>What sentiment does this text convey?</h3>  
<crowd-card>  
<div class="card">  
<Everything is great!>  
</div>  
</crowd-card>  
</div>  
  
<div class="right">  
<h3>Select an option</h3>  
  
<select name="sentiment2" style="font-size: large" required>  
<option value="">(Please select)</option>  
<option>Negative</option>  
<option>Neutral</option>  
<option>Positive</option>  
<option>Text is empty</option>  
</select>  
</div>  
</crowd-form>
```

## Attributes

The following attributes are supported by this element.

### heading

The text displayed at the top of the box.

### image

A URL to an image to be displayed within the box.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form](#) (p. 504)
- **Child elements:** none

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-checkbox

A UI component that can be checked or unchecked allowing a user to select multiple options from a set.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a Liquid template that uses the `<crowd-checkbox>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>

  <p>Find the official website for: <strong>{{ task.input.company }}</strong></p>
  <p>Do not give Yelp pages, LinkedIn pages, etc.</p>
  <p>Include the http:// prefix from the website</p>
  <crowd-input name="website" placeholder="http://example.com"></crowd-input>

  <crowd-checkbox name="website-found">Website Found</crowd-checkbox>

</crowd-form>
```

### Attributes

The following attributes are supported by this element.

#### checked

A Boolean switch that, if present, displays the check box as checked.

The following is an example of the syntax used to check a checkbox by default.

```
<crowd-checkbox name="checkbox" value="checked" checked>This box is checked</crowd-
checkbox>
```

#### disabled

A Boolean switch that, if present, displays the check box as disabled and prevents it from being checked.

The following is an example of the syntax used to disable a checkbox.

```
<crowd-checkbox name="disabledCheckBox" value="Disabled" disabled>Cannot be selected</
crowd-checkbox>
```

#### name

A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

#### required

A Boolean switch that, if present, requires the worker to provide input.

The following is an example of the syntax used to require a checkbox be selected.

```
<crowd-checkbox name="work_verified" required>Instructions were clear</crowd-checkbox>
```

### value

A string used as the name for the check box state in the output. Defaults to "on" if not specified.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** none

## Output

Provides a JSON object. The name string is the object name and the valuemstring is the property name for a Boolean value based on the check box state; true if checked, false if not checked.

### Example : Sample Element Outputs

#### Using the same name value for multiple boxes.

```
<!-- INPUT -->
<div><crowd-checkbox name="image_attributes" value="blurry"> Blurry </crowd-checkbox></div>
<div><crowd-checkbox name="image_attributes" value="dim"> Too Dim </crowd-checkbox></div>
<div><crowd-checkbox name="image_attributes" value="exposed"> Too Bright </crowd-
checkbox></div>
```

```
//Output with "blurry" and "dim" checked
[
  {
    "image_attributes": {
      "blurry": true,
      "dim": true,
      "exposed": false
    }
  }
]
```

Note that all three color values are properties of a single object.

#### Using different name values for each box.

```
<!-- INPUT -->
<div><crowd-checkbox name="Stop" value="Red"> Red </crowd-checkbox></div>
<div><crowd-checkbox name="Slow" value="Yellow"> Yellow </crowd-checkbox></div>
<div><crowd-checkbox name="Go" value="Green"> Green </crowd-checkbox></div>
```

```
//Output with "Red" checked
[
  {
    "Go": {
      "Green": false
    },
  }
]
```

```
        "Slow": {
            "Yellow": false
        },
        "Stop": {
            "Red": true
        }
    ]
}
```

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-classifier

A widget for classifying non-image content, such as audio, video, or text.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of an HTML worker task template built using `crowd-classifier`. This example uses the [Liquid template language](#) to automate:

- Label categories in the `categories` parameter
- The objects that are being classified in the `classification-target` parameter.

Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-classifier
    name="category"
    categories="{{ task.input.labels | to_json | escape }}"
    header="What type of a document is this?"
  >
    <classification-target>
      <iframe style="width: 100%; height: 600px;" src="{{ task.input.taskObject | grant_read_access }}" type="application/pdf"></iframe>
    </classification-target>

    <full-instructions header="Document Classification Instructions">
      <p>Read the task carefully and inspect the document.</p>
      <p>Choose the appropriate label that best suits the document.</p>
    </full-instructions>

    <short-instructions>
      Please choose the correct category for the document
    </short-instructions>
  </crowd-classifier>
</crowd-form>
```

## Attributes

The following attributes are supported by this element.

## categories

A JSON formatted array of strings, each of which is a category that a worker can assign to the text. You should include "other" as a category, otherwise the worker may not be able to provide an answer.

## header

The text to display above the image. This is typically a question or simple instruction for the worker.

## name

The name of this widget. It is used as a key for the widget's input in the form output.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** [classification-target \(p. 497\)](#), [full-instructions \(p. 497\)](#), [short-instructions \(p. 497\)](#)

## Regions

The following regions are supported by this element.

### classification-target

The content to be classified by the worker. This can be plain text or HTML. Examples of how the HTML can be used include *but are not limited to* embedding a video or audio player, embedding a PDF, or performing a comparison of two or more images.

### full-instructions

General instructions about how to do text classification.

### short-instructions

Important task-specific instructions that are displayed in a prominent place.

## Output

The output of this element is an object using the specified `name` value as a property name, and a string from the `categories` as the property's value.

### Example : Sample Element Outputs

The following is a sample of output from this element.

```
[  
  {  
    "<name>": {  
      "label": "<value>"  
    }  
  }  
]
```

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)

- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-classifier-multi-select

A widget for classifying various forms of content—such as audio, video, or text—into one or more categories. The content to classify is referred to as an *object*.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of an HTML worker task template built using this element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-classifier-multi-select
    name="category"
    categories="['Positive', 'Negative', 'Neutral']"
    header="Select the relevant categories"
    exclusion-category="{ text: 'None of the above' }"
  >
    <classification-target>
      {{ task.input.taskObject }}
    </classification-target>

    <full-instructions header="Text Categorization Instructions">
      <p><strong>Positive</strong> sentiment include: joy, excitement, delight</p>
      <p><strong>Negative</strong> sentiment include: anger, sarcasm, anxiety</p>
      <p><strong>Neutral</strong>: neither positive or negative, such as stating a fact</p>
      <p><strong>N/A</strong>: when the text cannot be understood</p>
      <p>When the sentiment is mixed, such as both joy and sadness, choose both labels.</p>
    </full-instructions>

    <short-instructions>
      Choose all categories that are expressed by the text.
    </short-instructions>
  </crowd-classifier-multi-select>
</crowd-form>
```

## Attributes

The following attributes are supported by the `crowd-classifier-multi-select` element. Each attribute accepts a string value or string values.

### categories

Required. A JSON-formatted array of strings, each of which is a category that a worker can assign to the object.

### header

Required. The text to display above the image. This is typically a question or simple instruction for workers.

### name

Required. The name of this widget. In the form output, the name is used as a key for the widget's input.

## exclusion-category

Optional. A JSON-formatted string with the following format: "{ text: '[default-value](#)' }". This attribute sets a default value that workers can choose if none of the labels applies to the object shown in the worker UI.

## Element Hierarchy

This element has the following parent and child elements:

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** [classification-target \(p. 497\)](#), [full-instructions \(p. 497\)](#), [short-instructions \(p. 497\)](#)

## Regions

This element uses the following regions.

### classification-target

The content to be classified by the worker. Content can be plain text or an object that you specify in the template using HTML. For example, you can use HTML elements to include a video or audio player, embedding a PDF file, or include a comparison of two or more images.

### full-instructions

General instructions about how to classify text.

### short-instructions

Important task-specific instructions. These instructions are displayed prominently.

## Output

The output of this element is an object that uses the specified name value as a property name, and a string from categories as the property's value.

### Example : Sample Element Outputs

The following is a sample of output from this element.

```
[  
  {  
    "<name>": {  
      labels: ["label_a", "label_b"]  
    }  
  }  
]
```

## See Also

For more information, see the following:

- [Text Classification \(Multi-label\) \(p. 194\)](#)
- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-entity-annotation

A widget for labeling words, phrases, or character strings within a longer text. Workers select a label, and highlight the text that the label applies to.

### Important: Self-contained Widget

Do not use `<crowd-entity-annotation>` element with the `<crowd-form>` element. It contains its own form submission logic and **Submit** button.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a template that uses the `<crowd-entity-annotation>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-entity-annotation
  name="crowd-entity-annotation"
  header="Highlight parts of the text below"
  labels="[{label: 'person', 'shortDisplayName': 'per', 'fullDisplayName': 'Person'},
  {'label': 'date', 'shortDisplayName': 'dat', 'fullDisplayName': 'Date'}, {'label':
  'company', 'shortDisplayName': 'com', 'fullDisplayName': 'Company'}]"
  text="Amazon SageMaker Ground Truth helps you build highly accurate training datasets for
  machine learning quickly."
>
  <full-instructions header="Named entity recognition instructions">
    <ol>
      <li><strong>Read</strong> the text carefully.</li>
      <li><strong>Highlight</strong> words, phrases, or sections of the text.</li>
      <li><strong>Choose</strong> the label that best matches what you have highlighted.</li>
      <li>To <strong>change</strong> a label, choose highlighted text and select a new
        label.</li>
        <li>To <strong>remove</strong> a label from highlighted text, choose the X next to
        the abbreviated label name on the highlighted text.</li>
        <li>You can select all of a previously highlighted text, but not a portion of it.</li>
    </ol>
  </full-instructions>

  <short-instructions>
    Apply labels to words or phrases.
  </short-instructions>

  <div id="additionalQuestions" style="margin-top: 20px">
    <h3>
      What is the overall subject of this text?
    </h3>
    <crowd-radio-group>
      <crowd-radio-button name="tech" value="tech">Technology</crowd-radio-button>
      <crowd-radio-button name="politics" value="politics">Politics</crowd-radio-button>
    </crowd-radio-group>
  </div>
</crowd-entity-annotation>

<script>
  document.addEventListener('all-crowd-elements-ready', () => {
    document
      .querySelector('crowd-entity-annotation')
      .shadowRoot
      .querySelector('crowd-form')
      .form
      .appendChild(additionalQuestions);
```

```
    });
</script>
```

## Attributes

The following attributes are supported by this element.

### header

The text to display above the image. This is typically a question or simple instruction for the worker.

### initial-value

A JSON formatted array of objects, each of which defines an annotation to apply to the text at initialization. Objects contain a `label` value that matches one in the `labels` attribute, an integer `startOffset` value for labeled span's starting unicode offset, and an integer `endOffset` value for the ending unicode offset.

## Example

```
[  
  {  
    label: 'person',  
    startOffset: 0,  
    endOffset: 16  
  },  
  ...  
]
```

### labels

A JSON formatted array of objects, each of which contains:

- **label** (required): The name used to identify entities.
- **fullDisplayName** (optional): Used for the label list in the task widget. Defaults to the label value if not specified.
- **shortDisplayName** (optional): An abbreviation of 3-4 letters to display above selected entities. Defaults to the label value if not specified.

#### **shortDisplayName is highly recommended**

Values displayed above the selections can overlap and create difficulty managing labeled entities in the workspace. Providing a 3-4 character `shortDisplayName` for each label is highly recommended to prevent overlap and keep the workspace manageable for your workers.

## Example

```
[  
  {  
    label: 'person',  
    shortDisplayName: 'per',  
    fullDisplayName: 'person'  
  }  
]
```

### name

Serves as the widget's name in the DOM. It is also used as the `label` attribute name in form output and the output manifest.

## text

The text to be annotated. The templating system escapes quotes and HTML strings by default. If your code is already escaped or partially escaped, see [Variable filters \(p. 305\)](#) for more ways to control escaping.

## Element Hierarchy

This element has the following parent and child elements.

- **Child elements:** [full-instructions \(p. 502\)](#), [short-instructions \(p. 502\)](#)

## Regions

The following regions are supported by this element.

### full-instructions

General instructions about how to work with the widget.

### short-instructions

Important task-specific instructions that are displayed in a prominent place.

## Output

The following output is supported by this element.

### entities

A JSON object that specifies the start, end, and label of an annotation. This object contains the following properties.

- **label** – The assigned label.
- **startOffset** – The Unicode offset of the beginning of the selected text.
- **endOffset** – The Unicode offset of the first character after the selection.

## Example : Sample Element Outputs

The following is a sample of the output from this element.

```
{  
  "myAnnotatedResult": {  
    "entities": [  
      {  
        "endOffset": 54,  
        "label": "person",  
        "startOffset": 47  
      },  
      {  
        "endOffset": 97,  
        "label": "event",  
        "startOffset": 93  
      },  
      {  
        "endOffset": 219,  
        "label": "date",  
        "startOffset": 212  
      },  
      {  
        "endOffset": 219,  
        "label": "date",  
        "startOffset": 212  
      }  
    ]  
  }  
}
```

```

        "endOffset": 271,
        "label": "location",
        "startOffset": 260
    }
]
}
}

```

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-fab

A floating button with an image in its center.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a Liquid template designed for image classification that uses the `<crowd-fab>` element. This template uses JavaScript to enable workers to report issues with the worker UI. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
    <crowd-image-classifier
        src="${image_url}"
        categories="['Cat', 'Dog', 'Bird', 'None of the Above']"
        header="Choose the correct category for the image"
        name="category">

        <short-instructions>
            <p>Read the task carefully and inspect the image.</p>
            <p>Choose the appropriate label that best suits the image.</p>
            <p>If there is an issue with the image or tools, please select
                <b>None of the Above</b>, describe the issue in the text box and click the
                button below.</p>
            <crowd-input label="Report an Issue" name="template-issues"></crowd-input>
            <crowd-fab id="button1" icon="report-problem" title="Issue"/>
        </short-instructions>

        <full-instructions header="Classification Instructions">
            <p>Read the task carefully and inspect the image.</p>
            <p>Choose the appropriate label that best suits the image.
                Use the <b>None of the Above</b> option if none of the other labels suit the
                image.</p>
        </full-instructions>

    </crowd-image-classifier>
</crowd-form>

<script>
    [
        button1,
    ].forEach(function(button) {
        button.addEventListener('click', function() {
            document.querySelector('crowd-form').submit();
    });

```

```
    });
});
</script>
```

## Attributes

The following attributes are supported by this element.

### disabled

A Boolean switch that, if present, displays the floating button as disabled and prevents clicks.

### icon

A string that specifies the icon to be displayed in the center of the button. The string must be either the name of an icon from the open-source [iron-icons](#) set, which is pre-loaded, or the URL to a custom icon.

The following is an example of the syntax that you can use to add an iron-icon to a `<crowd-fab>` HTML element. Replace `icon-name` with the name of the icon you'd like to use from this [Icons set](#).

```
<crowd-fab "id="button1" icon="icon-name" title="Issue"/>
```

### label

A string consisting of a single character that can be used instead of an icon. Emojis or multiple characters may result in the button displaying an ellipsis instead.

### title

A string that will display as a tool tip when the mouse hovers over the button.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** none

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-form

The form wrapper for all custom tasks. Sets and implements important actions for the proper submission of your form data.

If a [crowd-button \(p. 486\)](#) of type "submit" is not included inside the `<crowd-form>` element, it will automatically be appended within the `<crowd-form>` element.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of an image classification template that uses the `<crowd-form>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-image-classifier
    src="${image_url}"
    categories="['Cat', 'Dog', 'Bird', 'None of the Above']"
    header="Choose the correct category for the image"
    name="category">

    <short-instructions>
      <p>Read the task carefully and inspect the image.</p>
      <p>Choose the appropriate label that best suits the image.</p>
    </short-instructions>

    <full-instructions header="Classification Instructions">
      <p>Read the task carefully and inspect the image.</p>
      <p>Choose the appropriate label that best suits the image.
      Use the <b>None of the Above</b> option if none of the other labels suit the
      image.</p>
    </full-instructions>
  </crowd-image-classifier>
</crowd-form>
```

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** none
- **Child elements:** Any of the [UI Template \(p. 482\)](#) elements

## Element Events

The `crowd-form` element extends the [standard HTML form element](#) and inherits its events, such as `onclick` and `onsubmit`.

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-icon-button

A button with an image placed in the center. When the user touches the button, a ripple effect emanates from the center of the button.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a Liquid template designed for image classification that uses the `<crowd-icon-button>` element. This template uses JavaScript to enable workers to report issues with the worker UI. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
```

```

<crowd-form>
    <crowd-image-classifier
        src="${image_url}"
        categories="['Cat', 'Dog', 'Bird', 'None of the Above']"
        header="Choose the correct category for the image"
        name="category">

        <short-instructions>
            <p>Read the task carefully and inspect the image.</p>
            <p>Choose the appropriate label that best suits the image.</p>
            <p>If there is an issue with the image or tools, please select
                <b>None of the Above</b>, describe the issue in the text box and click the
                button below.</p>
            <crowd-input label="Report an Issue" name="template-issues"/></crowd-input>
            <crowd-icon-button id="button1" icon="report-problem" title="Issue"/>
        </short-instructions>

        <full-instructions header="Classification Instructions">
            <p>Read the task carefully and inspect the image.</p>
            <p>Choose the appropriate label that best suits the image.
                Use the <b>None of the Above</b> option if none of the other labels suit the
                image.</p>
        </full-instructions>

    </crowd-image-classifier>
</crowd-form>

<script>
    [
        button1,
    ].forEach(function(button) {
        button.addEventListener('click', function() {
            document.querySelector('crowd-form').submit();
        });
    });
</script>

```

## Attributes

The following attributes are supported by this element.

### disabled

A Boolean switch that, if present, displays the button as disabled and prevents clicks.

### icon

A string that specifies the icon to be displayed in the center of the button. The string must be either the name of an icon from the open-source [iron-icons](#) set, which is pre-loaded, or the URL to a custom icon.

The following is an example of the syntax that you can use to add an iron-icon to a `<crowd-icon-button>` HTML element. Replace `icon-name` with the name of the icon you'd like to use from this [Icons set](#).

```
<crowd-icon-button id="button1" icon="icon-name" title="Issue"/>
```

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)

- **Child elements:** none

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-image-classifier

A widget for classifying an image. Use one of the following supported image formats: APNG, BMP, GIF, ICO, JPEG, PNG, SVG. Images do not have a size limit.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of an image classification template that uses the `<crowd-image-classifier>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-image-classifier
    src="${image_url}"
    categories="['Cat', 'Dog', 'Bird', 'None of the Above']"
    header="Choose the correct category for the image"
    name="category">

    <short-instructions>
      <p>Read the task carefully and inspect the image.</p>
      <p>Choose the appropriate label that best suits the image.</p>
    </short-instructions>

    <full-instructions header="Classification Instructions">
      <p>Read the task carefully and inspect the image.</p>
      <p>Choose the appropriate label that best suits the image.
      Use the <b>None of the Above</b> option if none of the other labels suit the
      image.</p>
    </full-instructions>
  </crowd-image-classifier>
</crowd-form>
```

## Attributes

The following attributes are required by this element.

### categories

A JSON formatted array of strings, each of which is a category that a worker can assign to the image. You should include "other" as a category, so that the worker can provide an answer. You can specify up to 10 categories.

### header

The text to display above the image. This is typically a question or simple instruction for the worker.

### [name](#)

The name of this widget. It is used as a key for the widget's input in the form output.

### [overlay](#)

Information to be overlaid on the source image. This is for verification workflows of bounding-box, semantic-segmentation, and instance-segmentation tasks.

It is a JSON object containing an object with the name of the task-type in camelCase as the key. That key's value is an object that contains the labels and other necessary information from the previous task.

An example of a `crowd-image-classifier` element with attributes for verifying a bounding-box task follows:

```
<crowd-image-classifier
    name="boundingBoxClassification"
    header="Rate the quality of the annotations based on the background section
           in the instructions on the left hand side."
    src="https://i.imgur.com/CIPKVJo.jpg"
    categories="['good', 'bad', 'okay']"
    overlay='{
        "boundingBox": {
            labels: ["bird", "cat"],
            value: [
                {
                    height: 284,
                    label: "bird",
                    left: 230,
                    top: 974,
                    width: 223
                },
                {
                    height: 69,
                    label: "bird",
                    left: 79,
                    top: 889,
                    width: 247
                }
            ]
        },
    }'
> ... </crowd-image-classifier>
```

A semantic segmentation verification task would use the `overlay` value as follows:

```
<crowd-image-classifier
    name='crowd-image-classifier'
    categories=['good', 'bad']
    src='URL of image to be classified'
    header='Please classify'
    overlay='{
        "semanticSegmentation": {
            "labels": ["Cat", "Dog", "Bird", "Cow"],
            "labelMappings": {
                "Bird": {
                    "color": "#ff7f0e"
                },
                "Cat": {
                    "color": "#2ca02c"
                },
                "Cow": {
                    "color": "#d62728"
                }
            }
        }
    }'
```

```

        },
        "Dog": {
            "color": "#2acf59"
        }
    },
    "src": "URL of overlay image",
}
}'
> ... </crowd-image-classifier>

```

An instance-segmentation task would use the `overlay` value as follows:

```

<crowd-image-classifier
    name='crowd-image-classifier'
    categories='["good", "bad"]'
    src='URL of image to be classified'
    header='Please classify instances of each category'
    overlay='{
        "instanceSegmentation": {
            "labels": ["Cat", "Dog", "Bird", "Cow"],
            "instances": [
                {
                    "color": "#2ca02c",
                    "label": "Cat"
                },
                {
                    "color": "#1f77b4",
                    "label": "Cat"
                },
                {
                    "color": "#d62728",
                    "label": "Dog"
                }
            ],
            "src": "URL of overlay image",
        }
    }'
> ... </crowd-image-classifier>

```

## src

The URL of the image to be classified.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** [full-instructions \(p. 509\)](#), [short-instructions \(p. 509\)](#), [worker-comment \(p. 510\)](#)

## Regions

The following regions are used by this element.

### [full-instructions](#)

General instructions for the worker on how to classify an image.

### [short-instructions](#)

Important task-specific instructions that are displayed in a prominent place.

### worker-comment

Use this in verification workflows when you need workers to explain why they made the choice they did. Use the text between the opening and closing tags to provide instructions for workers on what information should be included in the comment.

It uses the following attributes:

#### header

A phrase with a call to action for leaving a comment. Used as the title text for a modal window where the comment is added.

Optional. Defaults to "Add a comment."

#### link-text

This text appears below the categories in the widget. When clicked, it opens a modal window where the worker may add a comment.

Optional. Defaults to "Add a comment."

#### placeholder

An example text in the comment text area that is overwritten when worker begins to type. This does not appear in output if the worker leaves the field blank.

Optional. Defaults to blank.

### Output

The output of this element is a string that specifies one of the values defined in the *categories* attribute of the <crowd-image-classifier> element.

### Example : Sample Element Outputs

The following is a sample of output from this element.

```
[  
  {  
    "<name>": {  
      "label": "<value>"  
      "workerComment": "Comment - if no comment is provided, this field will not be  
present"  
    }  
  }  
]
```

### See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-image-classifier-multi-select

A widget for classifying an image into one or more categories. Use one of the following supported image formats: APNG, BMP, GIF, ICO, JPEG, PNG, SVG. Images do not have a size limit.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of an HTML worker task template built using this crowd element. Copy the following code and save it in a file with the extension .html. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-image-classifier-multi-select
    name="animals"
    categories="['Cat', 'Dog', 'Horse', 'Pig', 'Bird']"
    src="https://images.unsplash.com/photo-1509205477838-a534e43a849f?ixlib=rb-1.2.1&ixid=eyJhcHBfaWQiOjEyMDd9&auto=format&fit=crop&w=1998&q=80"
    header="Please identify the animals in this image"
    exclusion-category="{ text: 'None of the above' }"
  >
    <full-instructions header="Classification Instructions">
      <p>If more than one label applies to the image, select multiple labels.</p>
      <p>If no labels apply, select <b>None of the above</b></p>
    </full-instructions>

    <short-instructions>
      <p>Read the task carefully and inspect the image.</p>
      <p>Choose the appropriate label(s) that best suit the image.</p>
    </short-instructions>
  </crowd-image-classifier-multi-select>
</crowd-form>
```

## Attributes

The following attributes are supported by the `crowd-image-classifier-multi-select` element. Each attribute accepts a string value or string values.

### categories

Required. A JSON-formatted array of strings, each of which is a category that a worker can assign to the image. A worker must choose at least one category and can choose all categories.

### header

Required. The text to display above the image. This is typically a question or simple instruction for workers.

### name

Required. The name of this widget. In the form output, the name is used as a key for the widget's input.

### src

Required. The URL of the image to be classified.

### exclusion-category

Optional. A JSON-formatted string with the following format: "{ text: '**default-value**' }". This attribute sets a default value that workers can choose if none of the labels applies to the image shown in the worker UI.

## Element Hierarchy

This element has the following parent and child elements:

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** [full-instructions \(p. 509\)](#), [short-instructions \(p. 509\)](#), [worker-comment \(p. 510\)](#)

## Regions

This element uses the following regions

### [full-instructions](#)

General instructions for the worker on how to classify an image.

### [short-instructions](#)

Important task-specific instructions. These instructions are displayed prominently.

## Output

The output of this element is a string that specifies one or more of the values defined in the categories attribute of the `<crowd-image-classifier-multi-select>` element.

### Example : Sample Element Outputs

The following is a sample of output from this element.

```
[  
  {  
    "<name>": {  
      labels: ["label_a", "label_b"]  
    }  
}
```

## See Also

For more information, see the following:

- [Image Classification \(Multi-label\) \(p. 183\)](#)
- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## [crowd-input](#)

A box that accepts input data.

### Cannot be self-closing

Unlike the `input` element in the HTML standard, this element cannot be self-closed by putting a slash before the ending bracket, e.g. `<crowd-input ... />`. It must be followed with a `</crowd-input>` to close the element.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a Liquid template that uses the `<crowd-input>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
```

```
<crowd-form>
  
  <crowd-input name="tag1" label="Word/phrase 1" required></crowd-input>
  <crowd-input name="tag2" label="Word/phrase 2" required></crowd-input>
  <crowd-input name="tag3" label="Word/phrase 3" required></crowd-input>

  <short-instructions>
    Your custom quick instructions and examples
  </short-instructions>

  <full-instructions>
    Your custom detailed instracutions and more examples
  </full-instructions>
</crowd-form>
```

## Attributes

The following attributes are supported by this element.

### allowed-pattern

A regular expression that is used with the *auto-validate* attribute to ignore non-matching characters as the worker types.

### auto-focus

When the value is set to true, the browser places focus inside the input area after loading. This way, the worker can start typing without having to select it first.

### auto-validate

A Boolean switch that, if present, turns on input validation. The behavior of the validator can be modified by the *error-message* and *allowed-pattern* attributes.

### disabled

A Boolean switch that, if present, displays the input area as disabled.

### error-message

The text to be displayed below the input field, on the left side, if validation fails.

### label

A string that is displayed inside a text field.

This text shrinks and rises up above a text field when the worker starts typing in the field or when the *value* attribute is set.

### max-length

A maximum number of characters the input will accept. Input beyond this limit is ignored.

### min-length

A minimum length for the input in the field

### name

Sets the name of the input to be used in the DOM and the output of the form.

### placeholder

A string value that is used as placeholder text, displayed until the worker starts entering data into the input. It is not used as a default value.

### required

A Boolean switch that, if present, requires the worker to provide input.

### type

Takes a string to set the HTML5 `input-type` behavior for the input. Examples include `file` and `date`.

### value

A preset that becomes the default if the worker does not provide input. The preset appears in a text field.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** none

## Output

Provides a `name` string as the property name, and the text that was entered in the field as its value.

### Example : Sample JSON Output

The values for multiple elements are output in the same object, with their `name` attribute value as their property name. Elements with no input do not appear in the output. For example, let's use three inputs:

```
<crowd-input name="tag1" label="Word/phrase 1"></crowd-input>
<crowd-input name="tag2" label="Word/phrase 2"></crowd-input>
<crowd-input name="tag3" label="Word/phrase 3"></crowd-input>
```

This is the output if only two have input:

```
[
  {
    "tag1": "blue",
    "tag2": "red"
  }
]
```

This means any code built to parse these results should be able to handle the presence or absence of each input in the answers.

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-instance-segmentation

A widget for identifying individual instances of specific objects within an image and creating a colored overlay for each labeled instance.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a Liquid template that uses the `<crowd-instance-segmentation>`. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-instance-segmentation
    name="annotatedResult"
    src="{{ task.input.taskObject | grant_read_access }}"
    header="Please label each of the requested objects in this image"
    labels="['Cat', 'Dog', 'Bird']"
  >
    <full-instructions header="Segmentation Instructions">
      <ol>
        <li><strong>Read</strong> the task carefully and inspect the image.</li>
        <li><strong>Read</strong> the options and review the examples provided to understand more about the labels.</li>
          <li><strong>Choose</strong> the appropriate label that best suits the image.</li>
        </ol>
    </full-instructions>

    <short-instructions>
      <p>Use the tools to label all instances of the requested items in the image</p>
    </short-instructions>
  </crowd-instance-segmentation>
</crowd-form>
```

Use a template similar to the following to allow workers to add their own categories (labels).

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-instance-segmentation
    id="annotator"
    name="myTexts"
    src="{{ task.input.taskObject | grant_read_access }}"
    header="Click Instructions to add new labels."
    labels="['placeholder']"
  >
    <short-instructions>
      <h3>Add a label to describe each type of object in this image.</h3>
      <h3>Cover each instance of each object with a segmentation mask.</h3>
      <br>
      <h3>
        Add new label
      </h3>
      <crowd-input name="_CustomLabel" id="CustomLabel"></crowd-input>
      <crowd-button id="addLabel">Add</crowd-button>
      <br><br><br>
      <h3>
        Manage labels
      </h3>
      <div id="labelsSection"></div>
    </short-instructions>
```

```

<full-instructions>
    Describe your task in more detail here.
</full-instructions>
</crowd-instance-segmentation>
</crowd-form>

<script>
    document.addEventListener('all-crowd-elements-ready', function(event) {
        document.querySelector('crowd-instance-segmentation').labels = [];
    });

    function populateLabelsSection() {
        labelsSection.innerHTML = '';
        annotator.labels.forEach(function(label) {
            const labelContainer = document.createElement('div');
            labelContainer.innerHTML = label + ' <a href="javascript:void(0)">(Delete)</a>';
            labelContainer.querySelector('a').onclick = function() {
                annotator.labels = annotator.labels.filter(function(l) {
                    return l !== label;
                });
                populateLabelsSection();
            };
            labelsSection.appendChild(labelContainer);
        });
    }

    addLabel.onclick = function() {
        annotator.labels = annotator.labels.concat([customLabel.value]);
        customLabel.value = null;

        populateLabelsSection();
    };
</script>

```

## Attributes

The following attributes are supported by this element.

### **header**

The text to display above the image. This is typically a question or simple instruction for the worker.

### **labels**

A JSON formatted array of strings, each of which is a label that a worker can assign to an instance of an object in the image. Workers can generate different overlay colors for each relevant instance by selecting "add instance" under the label in the tool.

### **name**

The name of this widget. It is used as a key for the labeling data in the form output.

### **src**

The URL of the image that is to be labeled.

### **initial-value**

A JSON object containing the color mappings of a prior instance segmentation job and a link to the overlay image output by the prior job. Include this when you want a human worker to verify the results of a prior labeling job and adjust it if necessary.

The attribute will appear as follows:

```
initial-value="{
  "instances": [
    {
      "color": "#2ca02c",
      "label": "Cat"
    },
    {
      "color": "#1f77b4",
      "label": "Cat"
    },
    {
      "color": "#d62728",
      "label": "Dog"
    }
  ],
  "src": {{ "S3 file URL for image" | grant_read_access }}
}"
```

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form](#) (p. 504)
- **Child elements:** [full-instructions](#) (p. 517), [short-instructions](#) (p. 517)

## Regions

The following regions are supported by this element.

### [full-instructions](#)

General instructions about how to do image segmentation.

### [short-instructions](#)

Important task-specific instructions that are displayed in a prominent place.

## Output

The following output is supported by this element.

### [labeledImage](#)

A JSON Object containing a Base64 encoded PNG of the labels.

### [instances](#)

A JSON Array containing objects with the instance labels and colors.

- **color** – The hexadecimal value of the label's RGB color in the `labeledImage` PNG.
- **label** – The label given to overlay(s) using that color. This value may repeat, because the different instances of the label are identified by their unique color.

### [inputImageProperties](#)

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.

- **width** – The width, in pixels, of the image.

### Example : Sample Element Outputs

The following is an example of output from this element.

```
[  
  {  
    "annotatedResult": {  
      "inputImageProperties": {  
        "height": 533,  
        "width": 800  
      },  
      "instances": [  
        {  
          "color": "#1f77b4",  
          "label": "<Label 1>":  
        },  
        {  
          "color": "#2ca02c",  
          "label": "<Label 1>":  
        },  
        {  
          "color": "#ff7f0e",  
          "label": "<Label 3>":  
        },  
      ],  
      "labeledImage": {  
        "pngImageData": "<Base-64 Encoded Data>"  
      }  
    }  
  }  
]
```

### See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-instructions

An element that displays instructions on three tabbed pages, **Summary**, **Detailed Instructions**, and **Examples**, when the worker clicks on a link or button.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a Liquid template that used the `<crowd-instructions>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>  
  
<crowd-form>  
  <crowd-instructions link-text="View instructions" link-type="button">  
    <short-summary>  
      <p>Given an image, write three words or short phrases that summarize its contents.</p>  
    </short-summary>  
  </crowd-instructions>  
</crowd-form>
```

```

</short-summary>
<detailed-instructions>
    <p>Imagine that you are describing an image to a friend or tagging it for a news website. Provide three specific words or short phrases that describe it.</p>
</detailed-instructions>
<positive-example>
    <p></p>
    <p>
        <ul>
            <li>Highway</li>
            <li>Cars</li>
            <li>Gas station</li>
        </ul>
    </p>
</positive-example>
<negative-example>
    <p></p>
    <p>
        These are not specific enough:
        <ol>
            <li>Trees</li>
            <li>Outside</li>
            <li>Daytime</li>
        </ol>
    </p>
</negative-example>
</crowd-instructions>
<p><strong>Instructions: </strong>Given an image, write three words or short phrases that summarize its contents.</p>
    <p>If someone were to see these three words or phrases, they should understand the subject and context of the image, as well as any important actions.</p>
    <p>View the instructions for detailed instructions and examples.</p>
    <p></p>
    <crowd-input name="tag1" label="Word/phrase 1" required></crowd-input>
    <crowd-input name="tag2" label="Word/phrase 2" required></crowd-input>
    <crowd-input name="tag3" label="Word/phrase 3" required></crowd-input>
</crowd-form>

```

## Attributes

The following attributes are supported by this element.

### link-text

The text to display for opening the instructions. The default is **Click for instructions**.

### link-type

A string that specifies the type of trigger for the instructions. The possible values are "link" (default) and "button".

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** none

## Regions

The following regions are supported by this element.

### [detailed-instructions](#)

Content that provides specific instructions for a task. This appears on the page of the "Detailed Instructions" tab.

### [negative-example](#)

Content that provides examples of inadequate task completion. This appears on the page of the "Examples" tab. More than one example may be provided within this element.

### [positive-example](#)

Content that provides examples of proper task completion. This appears on the page of the "Examples" tab.

### [short-summary](#)

A brief statement that summarizes the task to be completed. This appears on the page of the "Summary" tab. More than one example may be provided within this element.

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## [crowd-keypoint](#)

Generates a tool to select and annotate key points on an image.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of an Liquid template that uses the `<crowd-keypoint>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <div id="errorBox"></div>

  <crowd-keypoint
    src="{{ task.input.taskObject | grant_read_access }}"
    labels="['Item A', 'Item B', 'Item C']"
    header="Please locate the centers of each item."
    name="annotatedResult">
    <short-instructions>
      Describe your task briefly here and give examples
    </short-instructions>
    <full-instructions>
      Give additional instructions and good/bad examples here
    </full-instructions>
  </crowd-keypoint>
</crowd-form>

<script>
  var num_obj = 1;

  document.querySelector('crowd-form').onsubmit = function(e) {
```

```

const keypoints = document.querySelector('crowd-keypoint').value.keypoints ||
document.querySelector('crowd-keypoint')._submittableValue.keypoints;
const labels = keypoints.map(function(p) {
    return p.label;
});

// 1. Make sure total number of keypoints is correct.
var original_num_labels = document.getElementsByTagName("crowd-keypoint")
[0].getAttribute("labels");

original_num_labels = original_num_labels.substring(2, original_num_labels.length -
2).split("\",\"");
var goalNumKeypoints = num_obj*original_num_labels.length;
if (keypoints.length != goalNumKeypoints) {
    e.preventDefault();
    errorBox.innerHTML = '<crowd-alert type="error" dismissible>You must add all keypoint
annotations and use each label only once.</crowd-alert>';
    errorBox.scrollIntoView();
    return;
}

// 2. Make sure all labels are unique.
labelCounts = {};
for (var i = 0; i < labels.length; i++) {
    if (!labelCounts[labels[i]]) {
        labelCounts[labels[i]] = 0;
    }
    labelCounts[labels[i]]++;
}
const goalNumSingleLabel = num_obj;

const numLabels = Object.keys(labelCounts).length;

Object.entries(labelCounts).forEach(entry => {
    if (entry[1] != goalNumSingleLabel) {
        e.preventDefault();
        errorBox.innerHTML = '<crowd-alert type="error" dismissible>You must use each label
only once.</crowd-alert>';
        errorBox.scrollIntoView();
    }
});
};

</script>

```

## Attributes

The following attributes are supported by this element.

### header

The text to display above the image. This is typically a question or simple instruction for the worker.

### initial-value

An array, in JSON format, of keypoints to be applied to the image on start. For example:

```

initial-value="[
  {
    'label': 'Left Eye',
    'x': 1022,
    'y': 429
  },
  {
    'label': 'Beak',

```

```
        'x': 941,  
        'y': 403  
    }  
]
```

### Note

Please note that label values used in this attribute must have a matching value in the `labels` attribute or the point will not be rendered.

#### [labels](#)

An array, in JSON format, of strings to be used as keypoint annotation labels.

#### [name](#)

A string used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

#### [src](#)

The source URI of the image to be annotated.

### Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** [full-instructions \(p. 522\)](#), [short-instructions \(p. 522\)](#)

### Regions

The following regions are required by this element.

#### [full-instructions](#)

General instructions about how to annotate the image.

#### [short-instructions](#)

Important task-specific instructions that are displayed in a prominent place.

### Output

The following output is supported by this element.

#### [inputImageProperties](#)

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

#### [keypoints](#)

An array of JSON objects containing the coordinates and label of a keypoint. Each object contains the following properties.

- **label** – The assigned label for the keypoint.
- **x** – The X coordinate, in pixels, of the keypoint on the image.
- **y** – The Y coordinate, in pixels, of the keypoint on the image.

**Note**

X and Y coordinates are based on 0,0 being the top left corner of the image.

### Example : Sample Element Outputs

The following is a sample output from using this element.

```
[  
  {  
    "crowdKeypoint": {  
      "inputImageProperties": {  
        "height": 1314,  
        "width": 962  
      },  
      "keypoints": [  
        {  
          "label": "dog",  
          "x": 155,  
          "y": 275  
        },  
        {  
          "label": "cat",  
          "x": 341,  
          "y": 447  
        },  
        {  
          "label": "cat",  
          "x": 491,  
          "y": 513  
        },  
        {  
          "label": "dog",  
          "x": 714,  
          "y": 578  
        },  
        {  
          "label": "cat",  
          "x": 712,  
          "y": 763  
        },  
        {  
          "label": "cat",  
          "x": 397,  
          "y": 814  
        }  
      ]  
    }  
  ]
```

You may have many labels available, but only the ones that are used appear in the output.

### See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)

- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-line

A widget for drawing lines on an image. Each line is associated with a label, and output data will report the starting and ending points of each line.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a Liquid template that uses the `<crowd-line>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template. For more examples, see this [GitHub repository](#).

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-line
    name="crowdLine"
    src="{{ task.input.taskObject | grant_read_access }}"
    header="Add header here to describe the task"
    labels="['car','pedestrian','street car']"
  >
    <short-instructions>
      <p>Read the task carefully and inspect the image.</p>
      <p>Choose the appropriate label that best suits the image.</p>
      <p>Draw a line on each objects that the label applies to.</p>
    </short-instructions>

    <full-instructions>
      <p>Read the task carefully and inspect the image.</p>
      <p>Choose the appropriate label that best suits the image.
      <p>Draw a line along each object that the image applies to.
        Make sure that the line does not extend beyond the boundaries
        of the object.
      </p>
      <p>Each line is defined by a starting and ending point. Carefully
        place the starting and ending points on the boundaries of the object.</p>
    </full-instructions>

  </crowd-line>
</crowd-form>
```

## Attributes

The following attributes are supported by this element.

### header

Optional. The text to display above the image. This is typically a question or simple instruction for the worker.

### initial-value

Optional. An array of JSON objects, each of which sets a line when the component is loaded. Each JSON object in the array contains the following properties:

- **label** – The text assigned to the line as part of the labeling task. This text must match one of the labels defined in the `labels` attribute of the `<crowd-line>` element.
- **vertices** – the x and y pixel coordinates of the start point and end point of the line, relative to the top-left corner of the image.

```
initial-value="{
  lines: [
    {
      label: 'sideline', // label of this line annotation
      vertices:[           // an array of vertices which decide the position of the line
        {
          x: 84,
          y: 110
        },
        {
          x: 60,
          y: 100
        }
      ]
    },
    {
      label: 'yardline',
      vertices:[
        {
          x: 651,
          y: 498
        },
        {
          x: 862,
          y: 869
        }
      ]
    }
  ]
}"
```

Lines set via the `initial-value` property can be adjusted. Whether or not a worker answer was adjusted is tracked via an `initialValueModified` boolean in the worker answer output.

### Labels

Required. A JSON formatted array of strings, each of which is a label that a worker can assign to the line.

**Limit:** 10 labels

### label-colors

Optional. An array of strings. Each string is a hexadecimal (hex) code for a label.

### name

Required. The name of this widget. It's used as a key for the widget's input in the form output.

### src

Required. The URL of the image on which to draw lines.

## Regions

The following regions are required by this element.

### full-instructions

General instructions about how to draw lines.

### short-instructions

Important task-specific instructions that are displayed in a prominent place.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** [short-instructions \(p. 525\)](#), [full-instructions \(p. 525\)](#)

## Output

### inputImageProperties

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

### lines

A JSON Array containing objects with the line labels and vertices.

- **label** – The label given to a line.
- **vertices** – the x and y pixel coordinates of the start point and end point of the line, relative to the top-left corner of the image.

## Example : Sample Element Outputs

The following is an example of output from this element.

```
{
    "crowdLine": { //This is the name you set for the crowd-line
        "inputImageProperties": {
            "height": 1254,
            "width": 2048
        },
        "lines": [
            {
                "label": "yardline",
                "vertices": [
                    {
                        "x": 58,
                        "y": 295
                    },
                    {
                        "x": 1342,
                        "y": 398
                    }
                ]
            },
            {
                "label": "sideline",
                "vertices": [
                    {
                        "x": 472,
                        "y": 910
                    },
                    {
                        "x": 1480,
                        "y": 910
                    }
                ]
            }
        ]
    }
}
```

```
        "y": 600
    }
]
}
}
```

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-modal

A small window that pops up on the display when it is opened.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of the syntax that you can use with the `<crowd-modal>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-modal
link-text = "See Examples"
link-type = "button">
Example Modal Text</crowd-modal>
```

## Attributes

The following attributes are supported by this element.

### link-text

The text to display for opening the modal. The default is "Click to open modal".

### link-type

A string that specifies the type of trigger for the modal. The possible values are "link" (default) and "button".

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** none

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-polygon

A widget for drawing polygons on an image and assigning a label to the portion of the image that is enclosed in each polygon.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a Liquid template that uses the `<crowd-polygon>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-polygon
    name="annotatedResult"
    src="{{ task.input.taskObject | grant_read_access }}"
    header="Draw a polygon around each of the requested target(s) of interest"
    labels="['Cat', 'Dog', 'Bird']"
  >
    <full-instructions header="Polygon instructions">
      <ul>
        <li>Make the polygon tight around the object</li>
        <li>You need to select a label before starting a polygon</li>
        <li>You will need to select a label again after completing a polygon</li>
        <li>To select a polygon, you can click on its borders</li>
        <li>You can start drawing a polygon from inside another polygon</li>
        <li>You can undo and redo while you're drawing a polygon to go back and forth between points you've placed</li>
        <li>You are prevented from drawing lines that overlap other lines from the same polygon</li>
      </ul>
    </full-instructions>

    <short-instructions>
      <p>Draw a polygon around each of the requested target(s) of interest</p>
      <p>Make the polygon tight around the object</p>
    </short-instructions>
  </crowd-polygon>
</crowd-form>
```

## Attributes

The following attributes are supported by this element.

### header

The text to display above the image. This is typically a question or simple instruction for the worker.

### labels

A JSON formatted array of strings, each of which is a label that a worker can assign to the image portion enclosed by a polygon.

### name

The name of this widget. It's used as a key for the widget's input in the form output.

### src

The URL of the image on which to draw polygons.

### initial-value

An array of JSON objects, each of which defines a polygon to be drawn when the component is loaded. Each JSON object in the array contains the following properties.

- **label** – The text assigned to the polygon as part of the labeling task. This text must match one of the labels defined in the *labels* attribute of the `<crowd-polygon>` element.
- **vertices** – An array of JSON objects. Each object contains an x and y coordinate value for a point in the polygon.

### Example

An `initial-value` attribute might look something like this.

```
initial-value =  
  '[  
    [  
      {  
        "label": "dog",  
        "vertices":  
          [  
            {  
              "x": 570,  
              "y": 239  
            },  
            ...  
            {  
              "x": 759,  
              "y": 281  
            }  
          ]  
      }  
    ]'
```

Because this will be within an HTML element, the JSON array must be enclosed in single or double quotes. The example above uses single quotes to encapsulate the JSON and double quotes within the JSON itself. If you must mix single and double quotes inside your JSON, replace them with their HTML entity codes (`&quot;` for double quote, `&#39;` for single) to safely escape them.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** [full-instructions \(p. 529\)](#), [short-instructions \(p. 529\)](#)

## Regions

The following regions are required.

### full-instructions

General instructions about how to draw polygons.

### short-instructions

Important task-specific instructions that are displayed in a prominent place.

## Output

The following output is supported by this element.

### polygons

An array of JSON objects, each of which describes a polygon that has been created by the worker. Each JSON object in the array contains the following properties.

- **label** – The text assigned to the polygon as part of the labeling task.
- **vertices** – An array of JSON objects. Each object contains an x and y coordinate value for a point in the polygon. The top left corner of the image is 0,0.

### inputImageProperties

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

## Example : Sample Element Outputs

The following are samples of outputs from common use scenarios for this element.

### Single Label, Single Polygon

```
{  
    "annotatedResult":  
    {  
        "inputImageProperties": {  
            "height": 853,  
            "width": 1280  
        },  
        "polygons":  
        [  
            {  
                "label": "dog",  
                "vertices":  
                [  
                    {  
                        "x": 570,  
                        "y": 239  
                    },  
                    {  
                        "x": 603,  
                        "y": 513  
                    },  
                    {  
                        "x": 823,  
                        "y": 645  
                    },  
                    {  
                        "x": 901,  
                        "y": 417  
                    },  
                    {  
                        "x": 759,  
                        "y": 281  
                    }  
                ]  
            }  
        ]  
    }  
}
```

```
        ]
    }
]
```

### Single Label, Multiple Polygons

```
[
  {
    "annotatedResult": {
      "inputImageProperties": {
        "height": 853,
        "width": 1280
      },
      "polygons": [
        {
          "label": "dog",
          "vertices": [
            {
              "x": 570,
              "y": 239
            },
            {
              "x": 603,
              "y": 513
            },
            {
              "x": 823,
              "y": 645
            },
            {
              "x": 901,
              "y": 417
            },
            {
              "x": 759,
              "y": 281
            }
          ]
        },
        {
          "label": "dog",
          "vertices": [
            {
              "x": 870,
              "y": 278
            },
            {
              "x": 908,
              "y": 446
            },
            {
              "x": 1009,
              "y": 602
            },
            {
              "x": 1116,
              "y": 519
            },
            {
              "x": 1174,
              "y": 498
            }
          ]
        }
      ]
    }
]
```

```
        "x": 1227,
        "y": 479
    },
    {
        "x": 1179,
        "y": 405
    },
    {
        "x": 1179,
        "y": 337
    }
]
}
]
```

### Multiple Labels, Multiple Polygons

```
[
{
    "annotatedResult": {
        "inputImageProperties": {
            "height": 853,
            "width": 1280
        },
        "polygons": [
            {
                "label": "dog",
                "vertices": [
                    {
                        "x": 570,
                        "y": 239
                    },
                    {
                        "x": 603,
                        "y": 513
                    },
                    {
                        "x": 823,
                        "y": 645
                    },
                    {
                        "x": 901,
                        "y": 417
                    },
                    {
                        "x": 759,
                        "y": 281
                    }
                ]
            },
            {
                "label": "cat",
                "vertices": [
                    {
                        "x": 870,
                        "y": 278
                    },
                    {
                        "x": 908,
                        "y": 446
                    }
                ]
            }
        ]
    }
}
```

```
        "x": 1009,
        "y": 602
    },
    {
        "x": 1116,
        "y": 519
    },
    {
        "x": 1174,
        "y": 498
    },
    {
        "x": 1227,
        "y": 479
    },
    {
        "x": 1179,
        "y": 405
    },
    {
        "x": 1179,
        "y": 337
    }
]
}
]
```

You could have many labels available, but only the ones that are used appear in the output.

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-polyline

A widget for drawing polylines or lines on an image. Each polyline is associated with a label and can include two or more vertices. A polyline can intersect itself and its starting and ending points can be placed anywhere on the image.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a Liquid template that uses the `<crowd-polyline>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template. For more examples, see this [GitHub repository](#).

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-polyline
    name="crowdPolyline"
    src="{{ task.input.taskObject | grant_read_access }}"
    header="Add header here to describe the task"
    labels="['car','pedestrian','street car']"
  >
    <full-instructions>
```

```

<p>Read the task carefully and inspect the image.</p>
<p>Choose the appropriate label that best suits the image.</p>
<p>Draw a polyline around the boundaries of all objects
that the label applies to.</p>
<p>Use the <b>Enter</b> key to complete a polyline.</p>
<p>Make sure that the polyline fits tightly around the boundary
of the object.</p>
</full-instructions>

<short-instructions>
    <p>Read the task carefully and inspect the image.</p>
    <p>Review the tool guide to learn how to use the polyline tool.</p>
    <p>Choose the appropriate label that best suits the image.</p>
    <p>To draw a polyline, select a label that applies to an object of interest
        and add a single point to the photo by clicking on that point. Continue to
        draw the polyline around the object by adding additional points
        around the object boundary.</p>
    <p>After you place the final point on the polyline, press <b>Enter</b> on your
        keyboard to complete the polyline.</p>
</short-instructions>
</crowd-polyline>
</crowd-form>

```

## Attributes

The following attributes are supported by this element.

### header

Optional. The text to display above the image. This is typically a question or simple instruction for the worker.

### initial-value

Optional. An array of JSON objects, each of which sets a polyline when the component is loaded. Each JSON object in the array contains the following properties:

- **label** – The text assigned to the polyline as part of the labeling task. This text must match one of the labels defined in the *labels* attribute of the `<crowd-polyline>` element.
- **vertices** – the x and y pixel coordinates of the vertices of a polyline, relative to the top-left corner of the image.

```

initial-value= "{
    polylines: [
        {
            label: 'sideline', // label of this line annotation
            vertices:[           // an array of vertices which decide the position of the line
                {
                    x: 84,
                    y: 110
                },
                {
                    x: 60,
                    y: 100
                }
            ]
        },
        {
            label: 'yardline',
            vertices:[
                {

```

```
        x: 651,
        y: 498
    },
    {
        x: 862,
        y: 869
    },
    {
        x: 1000,
        y: 869
    }
]
}"
```

Polyline segments set via the `initial-value` property can be adjusted. Whether or not a worker answer was adjusted is tracked via an `initialValueModified` boolean in the worker answer output.

### labels

Required. A JSON formatted array of strings, each of which is a label that a worker can assign to the line.

**Limit:** 10 labels

### label-colors

Optional. An array of strings. Each string is a hexadecimal (hex) code for a label.

### name

Required. The name of this widget. It's used as a key for the widget's input in the form output.

### src

Required. The URL of the image on which to draw polylines.

## Regions

The following regions are required by this element.

### full-instructions

General instructions about how to draw polylines.

### short-instructions

Important task-specific instructions that are displayed in a prominent place.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** [short-instructions \(p. 535\)](#), [full-instructions \(p. 535\)](#)

## Output

### inputImageProperties

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

## polylines

A JSON Array containing objects with polylines' labels and vertices.

- **label** – The label given to a line.
- **vertices** – the x and y pixel coordinates of the vertices of a polyline, relative to the top-left corner of the image.

## Example : Sample Element Outputs

The following is an example of output from this element.

```
{
    "crowdPolyline": { //This is the name you set for the crowd-polyline
        "inputImageProperties": {
            "height": 1254,
            "width": 2048
        },
        "polylines": [
            {
                "label": "sideline",
                "vertices": [
                    {
                        "x": 651,
                        "y": 498
                    },
                    {
                        "x": 862,
                        "y": 869
                    },
                    {
                        "x": 1449,
                        "y": 611
                    }
                ]
            },
            {
                "label": "yardline",
                "vertices": [
                    {
                        "x": 1148,
                        "y": 322
                    },
                    {
                        "x": 1705,
                        "y": 474
                    },
                    {
                        "x": 1755,
                        "y": 474
                    }
                ]
            }
        ]
    }
}
```

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-radio-button

A button that can be either checked or unchecked. When radio buttons are inside a radio group, exactly one radio button in the group can be checked at any time. The following is an example of how to configure a `crowd-radio-button` element inside of a `crowd-radio-group` element.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of the syntax that you can use with the `<crowd-radio-button>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
<crowd-radio-group>
    <crowd-radio-button name="tech" value="tech">Technology</crowd-radio-button>
    <crowd-radio-button name="politics" value="politics">Politics</crowd-radio-button>
</crowd-form>
</crowd-radio-group>
```

The previous example can be seen in a custom worker task template in this GitHub example: [entity recognition labeling job custom template](#).

Crowd HTML Element radio buttons do not support the `HTML` tag, required. To make a radio button selection required, use `<input type="radio">` elements to create radio buttons and add the `required` tag. The `name` attribute for all `<input>` elements that belong to the same group of radio buttons must be the same. For example, the following template requires the user to select a radio button in the `animal-type` group before submitting.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
    <p>Select an animal type:</p>
    
    <br><br>
    <div>
        <input type="radio" id="cat" name="animal-type" value="cat" required>
        <label for="cat">Cat</label>
    </div>
    <div>
        <input type="radio" id="dog" name="animal-type" value="dog">
        <label for="dog">Dog</label>
    </div>
    <div>
        <input type="radio" id="unknown" name="animal-type" value="unknown">
        <label for="unknown">Unknown</label>
    </div>
    <full-instructions header="Classification Instructions">
        <p>Read the task carefully and inspect the image.</p>
        <p>Choose the appropriate label that best suits the image.</p>
    </full-instructions>
    <short-instructions>
```

```
<p>Read the task carefully and inspect the image.</p>
<p>Choose the appropriate label that best suits the image.</p>
</short-instructions>
</crowd-form>
```

## Attributes

The following attributes are supported by this element.

### checked

A Boolean switch that, if present, displays the radio button as checked.

### disabled

A Boolean switch that, if present, displays the button as disabled and prevents it from being checked.

### name

A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

#### Note

If you use the buttons outside of a [crowd-radio-group \(p. 539\)](#) element, but with the same name string and different value strings, the name object in the output will contain a Boolean value for each value string. To ensure that only one button in a group is selected, make them children of a [crowd-radio-group \(p. 539\)](#) element and use different name values.

### value

A property name for the element's boolean value. If not specified, it uses "on" as the default, e.g.  
{ "<name>": { "<value>": <true or false> } }.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-radio-group \(p. 539\)](#)
- **Child elements:** none

## Output

Outputs an object with the following pattern: { "<name>": { "<value>": <true or false> } }. If you use the buttons outside of a [crowd-radio-group \(p. 539\)](#) element, but with the same name string and different value strings, the name object will contain a Boolean value for each value string. To ensure that only one in a group of buttons is selected, make them children of a [crowd-radio-group \(p. 539\)](#) element and use different name values.

## Example Sample output of this element

```
[  
  {  
    "btn1": {  
      "yes": true  
    },  
    "btn2": {  
      "no": false  
    }  
}
```

]

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-radio-group

A group of radio buttons. Only one radio button within the group can be selected. Choosing one radio button clears any previously chosen radio button within the same group. For an example of a custom UI template that uses the `crowd-radio-group` element, see this [entity recognition labeling job custom template](#).

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of the syntax that you can use with the `<crowd-radio-group>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<style>
body {
  padding-left: 20px;
  margin-bottom: 20px;
}
#outer-container {
  display: flex;
  justify-content: space-around;
  max-width: 900px;
  margin-left: 100px;
}
.left-container {
  margin-right: auto;
  padding-right: 50px;
}
.right-container {
  margin-left: auto;
  padding-left: 50px;
}
#vertical-separator {
  border: solid 1px #d5dbdb;
}
</style>

<crowd-form>
  <div>
    <h1>Instructions</h1>
    Lorem ipsum...
  </div>
  <div>
    <h2>Background</h2>
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
    incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud
    exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.</p>
  </div>
<div id="outer-container">
```

```

<span class="left-container">
    <h2>Option 1</h2>
    <p>Nulla facilisi morbi tempus iaculis urna. Orci dapibus ultrices in iaculis nunc sed
    augue lacus.</p>
</span>
<span id="vertical-separator"></span>
<span class="right-container">
    <h2>Option 2</h2>
    <p>Ultrices vitae auctor eu augue ut. Pellentesque massa placerat dui ultricies lacus
    sed turpis tincidunt id.</p>
</span>
</div>
<div>
    <h2>Question</h2>
    <p>Which do you agree with?</p>
<crowd-radio-group>
    <crowd-radio-button name="option1" value="Option 1">Option 1</crowd-radio-button>
    <crowd-radio-button name="option2" value="Option 2">Option 2</crowd-radio-button>
</crowd-radio-group>

    <p>Why did you choose this answer?</p>
<crowd-text-area name="explanation" placeholder="Explain how you reached your
conclusion..."></crowd-text-area>
</div>
</crowd-form>

```

## Attributes

No special attributes are supported by this element.

## Element Hierarchy

This element has the following parent and child elements.

- Parent elements:** [crowd-form \(p. 504\)](#)
- Child elements:** [crowd-radio-button \(p. 537\)](#)

## Output

Outputs an array of objects representing the [crowd-radio-button \(p. 537\)](#) elements within it.

### Example Sample of Element Output

```
[
  {
    "btn1": {
      "yes": true
    },
    "btn2": {
      "no": false
    }
  }
]
```

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-semantic-segmentation

A widget for segmenting an image and assigning a label to each image segment.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a Liquid template that uses the `<crowd-semantic-segmentation>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-semantic-segmentation
    name="annotatedResult"
    src="{{ task.input.taskObject | grant_read_access }}"
    header="Please label each of the requested objects in this image"
    labels="['Cat', 'Dog', 'Bird']"
  >
    <full-instructions header="Segmentation Instructions">
      <ol>
        <li><strong>Read</strong> the task carefully and inspect the image.</li>
        <li><strong>Read</strong> the options and review the examples provided to understand more about the labels.</li>
        <li><strong>Choose</strong> the appropriate label that best suits the image.</li>
      </ol>
    </full-instructions>

    <short-instructions>
      <p>Use the tools to label the requested items in the image</p>
    </short-instructions>
  </crowd-semantic-segmentation>
</crowd-form>
```

## Attributes

The following attributes are supported by this element.

### header

The text to display above the image. This is typically a question or simple instruction for the worker.

### initial-value

A JSON object containing the color mappings of a prior semantic segmentation job and a link to the overlay image output by the prior job. Include this when you want a human worker to verify the results of a prior labeling job and adjust it if necessary.

The attribute would appear as follows:

```
initial-value='{
  "labelMappings": {
    "Bird": {
      "color": "#ff7f0e"
    },
    "Cat": {
      "color": "#2ca02c"
    },
    "Cow": {
      "color": "#d62728"
    },
  }
},'
```

```

        "Dog": {
            "color": "#1f77b4"
        }
    },
    "src": {{ "S3 file URL for image" | grant_read_access }}
}"

```

When using Ground Truth [built in task types](#) with [annotation consolidation](#) (where more than one worker labels a single image), label mappings are included in individual worker output records, however the overall result is represented as the `internal-color-map` in the consolidated results.

You can convert the `internal-color-map` to `label-mappings` in a custom template using the Liquid templating language:

```

initial-value="{
    'src' : '{{ task.input.manifestLine.label-attribute-name-from-prior-job |
    grant_read_access }}',
    'labelMappings': {
        {% for box in task.input.manifestLine.label-attribute-name-from-prior-job-
metadata.internal-color-map %}
            {% if box[1]['class-name'] != 'BACKGROUND' %}
                {{ box[1]['class-name'] | to_json }}: {
                    'color': {{ box[1]['hex-color'] | to_json }}
                },
                {% endif %}
            {% endfor %}
    }
}"

```

## labels

A JSON formatted array of strings, each of which is a label that a worker can assign to a segment of the image.

## name

The name of this widget. It is used as a key for the widget's input in the form output.

## src

The URL of the image that is to be segmented.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** [full-instructions \(p. 542\)](#), [short-instructions \(p. 542\)](#)

## Regions

The following regions are supported by this element.

### full-instructions

General instructions about how to do image segmentation.

### short-instructions

Important task-specific instructions that are displayed in a prominent place.

## Output

The following output is supported by this element.

### `labeledImage`

A JSON Object containing a Base64 encoded PNG of the labels.

### `labelMappings`

A JSON Object containing objects with named with the segmentation labels.

- **color** – The hexadecimal value of the label's RGB color in the `labeledImage` PNG.

### `initialValueModified`

A boolean representing whether the initial values have been modified. This is only included when the output is from an adjustment task.

### `inputImageProperties`

A JSON object that specifies the dimensions of the image that is being annotated by the worker. This object contains the following properties.

- **height** – The height, in pixels, of the image.
- **width** – The width, in pixels, of the image.

## Example : Sample Element Outputs

The following is a sample of output from this element.

```
[  
  {  
    "annotatedResult": {  
      "inputImageProperties": {  
        "height": 533,  
        "width": 800  
      },  
      "labelMappings": {  
        "<Label 2>": {  
          "color": "#ff7f0e"  
        },  
        "<label 3>": {  
          "color": "#2ca02c"  
        },  
        "<label 1>": {  
          "color": "#1f77b4"  
        }  
      },  
      "labeledImage": {  
        "pngImageData": "<Base-64 Encoded Data>"  
      }  
    }  
  }  
]
```

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-slider

A bar with a sliding knob that allows a worker to select a value from a range of values by moving the knob. The slider makes it a great choice for settings that reflect intensity levels, such as volume, brightness, or color saturation.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a survey template that uses the `<crowd-slider>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-instructions link-text="View instructions" link-type="button">
    <short-summary>
      <p>Provide a brief instruction here</p>
    </short-summary>

    <detailed-instructions>
      <h3>Provide more detailed instructions here</h3>
      <p>Include additional information</p>
    </detailed-instructions>

    <positive-example>
      <p>Provide an example of a good answer here</p>
      <p>Explain why it's a good answer</p>
    </positive-example>

    <negative-example>
      <p>Provide an example of a bad answer here</p>
      <p>Explain why it's a bad answer</p>
    </negative-example>
  </crowd-instructions>

  <div>
    <p>What is your favorite color for a bird?</p>
    <crowd-input name="favoriteColor" placeholder="example: pink" required></crowd-input>
  </div>

  <div>
    <p>Check this box if you like birds</p>
    <crowd-checkbox name="likeBirds" checked="true" required></crowd-checkbox>
  </div>

  <div>
    <p>On a scale of 1-10, how much do you like birds?</p>
    <crowd-slider name="howMuch" min="1" max="10" step="1" pin="true" required></crowd-slider>
  </div>

  <div>
    <p>Write a short essay describing your favorite bird</p>
    <crowd-text-area name="essay" rows="4" placeholder="Lorem ipsum..." required></crowd-text-area>
  </div>
</crowd-form>
```

## Attributes

The following attributes are supported by this element.

### `disabled`

A Boolean switch that, if present, displays the slider as disabled.

### `editable`

A Boolean switch that, if present, displays an up/down button that can be chosen to select the value.

Selecting the value via the up/down button is an alternative to selecting the value by moving the knob on the slider. The knob on the slider will move synchronously with the up/down button choices.

### `max`

A number that specifies the maximum value on the slider.

### `min`

A number that specifies the minimum value on the slider.

### `name`

A string that is used to identify the answer submitted by the worker. This value will match a key in the JSON object that specifies the answer.

### `pin`

A Boolean switch that, if present, displays the current value above the knob as the knob is moved.

### `required`

A Boolean switch that, if present, requires the worker to provide input.

### `secondary-progress`

When used with a `crowd-slider-secondary-color` CSS attribute, the progress bar is colored to the point represented by the `secondary-progress`. For example, if this was representing the progress on a streaming video, the value would represent where the viewer was in the video timeline. The `secondary-progress` value would represent the point on the timeline to which the video had buffered.

### `step`

A number that specifies the difference between selectable values on the slider.

### `value`

A preset that becomes the default if the worker does not provide input.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form](#) (p. 504)
- **Child elements:** none

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-tab

A component styled to look like a tab with information below.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example template that uses the `<crowd-tab>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-tabs>
    <crowd-tab header="Tab 1">
      <h2>Image</h2>

      <h2>Text</h2>
      <p>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
        incididunt ut labore et dolore magna aliqua.
      </p>
      <p>
        Sed risus ultricies tristique nulla aliquet enim tortor at auctor. Tempus egestas sed
        sed risus.
      </p>
    </crowd-tab>

    <crowd-tab header="Tab 2">
      <h2>Description</h2>
      <p>
        Sed risus ultricies tristique nulla aliquet enim tortor at auctor. Tempus egestas sed
        sed risus.
      </p>
    </crowd-tab>

    <crowd-tab header="Tab 3">
      <div style="width: 40%; display: inline-block">
        
        <crowd-input label="Input inside tab" name="inputInsideTab"></crowd-input>
        <input type="checkbox" name="checkbox" value="foo">Foo
        <input type="checkbox" name="checkbox" value="bar">Bar
        <crowd-button>Some button</crowd-button>
      </div>
    </crowd-tab>
  </crowd-tabs>
</crowd-form>
```

```
<div style="width: 40%; display: inline-block; vertical-align: top">
    Lorem ipsum dolor sit amet, lorem a wisi nibh, in pulvinar, consequat praesent
    vestibulum tellus ante felis auctor, vitae lobortis dictumst mauris.
    Pellentesque nulla ipsum ante quisque quam augue.
    Class lacus id euismod, blandit tempor mauris quisque tortor mauris, urna gravida
    nullam pede libero, ut suscipit orci faucibus lacus varius ornare, pellentesque ipsum.
    At etiam suspendisse est elementum luctus netus, vel sem nulla sodales, potenti
    magna enim ipsum diam tortor rutrum,
    quam donec massa elit ac, nam adipiscing sed at leo ipsum consectetur. Ac turpis
    amet wisi, porttitor sint lacus ante, turpis accusantium, ac maecenas deleniti,
    nisl leo sem integer ac dignissim. Lobortis etiam luctus lectus odio auctor. Justo
    vitae, felis integer id, bibendum accumsan turpis eu est mus eros, ante id eros.
</div>
</crowd-tab>

</crowd-tabs>

<crowd-input label="Input outside tabs" name="inputOutsideTab"></crowd-input>

<short-instructions>
    <p>Sed risus ultricies tristique nulla aliquet enim tortor at auctor. Tempus egestas
    sed sed risus.</p>
</short-instructions>

<full-instructions header="Classification Instructions">
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
    incididunt ut labore et dolore magna aliqua.</p>
    <p> Tempus egestas sed sed risus.</p>
</full-instructions>

</crowd-form>
```

## Attributes

The following attributes are supported by this element.

### header

The text appearing on the tab. This is usually some short descriptive name indicative of the information contained below the tab.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-tabs \(p. 547\)](#)
- **Child elements:** none

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-tabs

A container for tabbed information.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example template that uses the `<crowd-tabs>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <crowd-tabs>
    <crowd-tab header="Tab 1">
      <h2>Image</h2>

      <h2>Text</h2>
      <p>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor
        incididunt ut labore et dolore magna aliqua.
      </p>
      <p>
        Sed risus ultricies tristique nulla aliquet enim tortor at auctor. Tempus egestas sed
        sed risus.
      </p>
    </crowd-tab>

    <crowd-tab header="Tab 2">
      <h2>Description</h2>
      <p>
        Sed risus ultricies tristique nulla aliquet enim tortor at auctor. Tempus egestas sed
        sed risus.
      </p>
    </crowd-tab>

    <crowd-tab header="Tab 3">
      <div style="width: 40%; display: inline-block">
        
        <crowd-input label="Input inside tab" name="inputInsideTab"></crowd-input>
        <input type="checkbox" name="checkbox" value="foo">Foo
        <input type="checkbox" name="checkbox" value="bar">Bar
        <crowd-button>Some button</crowd-button>
      </div>

      <div style="width: 40%; display: inline-block; vertical-align: top">
        Lorem ipsum dolor sit amet, lorem a wisi nibh, in pulvinar, consequat praesent
        vestibulum tellus ante felis auctor, vitae lobortis dictumst mauris.
        Pellentesque nulla ipsum ante quisque quam augue.
        Class lacus id euismod, blandit tempor mauris quisque tortor mauris, urna gravida
        nullam pede libero, ut suscipit orci faucibus lacus varius ornare, pellentesque ipsum.
        At etiam suspendisse est elementum luctus netus, vel sem nulla sodales, potenti
        magna enim ipsum diam tortor rutrum,
        quam donec massa elit ac, nam adipiscing sed at leo ipsum consectetur.
        Ac turpis
        amet wisi, porttitor sint lacus ante, turpis accusantium, ac maecenas deleniti,
        nisl leo sem integer ac dignissim. Lobortis etiam luctus lectus odio auctor. Justo
        vitae, felis integer id, bibendum accumsan turpis eu est mus eros, ante id eros.
      </div>
    </crowd-tab>
  </crowd-tabs>
</crowd-form>

```

```
</crowd-tabs>

<crowd-input label="Input outside tabs" name="inputOutsideTab"></crowd-input>

<short-instructions>
    <p>Sed risus ultricies tristique nulla aliquet enim tortor at auctor. Tempus egestas sed sed risus.</p>
</short-instructions>

<full-instructions header="Classification Instructions">
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.</p>
    <p>Tempus egestas sed sed risus.</p>
</full-instructions>

</crowd-form>
```

## Attributes

This element has no attributes.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** [crowd-tab \(p. 546\)](#)

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-text-area

A field for text input.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a Liquid template designed to transcribe audio clips that uses the `<crowd-text-area>` element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
    <audio controls>
        <source src="{{ task.input.taskObject | grant_read_access }}" type="audio/mpeg">
        Your browser does not support the audio element.
    </audio>
    <h3>Instructions</h3>
    <p>Transcribe the audio</p>
    <p>Ignore "umms", "hmms", "uhs" and other non-textual phrases</p>
    <crowd-text-area name="transcription" rows="4"></crowd-text-area>
</crowd-form>
```

## Attributes

The following attributes are supported by this element.

### auto-focus

A Boolean switch that, if present, puts the cursor in this element on-load so that users can immediately begin typing without having to click inside the element.

### auto-validate

A Boolean switch that, if present, turns on input validation. The behavior of the validator can be modified by the *error-message* and *allowed-pattern* attributes.

### char-counter

A Boolean switch that, if present, puts a small text field beneath the lower-right corner of the element, displaying the number of characters inside the element.

### disabled

A Boolean switch that, if present, displays the input area as disabled.

### error-message

The text to be displayed below the input field, on the left side, if validation fails.

### label

A string that is displayed inside a text field.

This text shrinks and rises up above a text field when the worker starts typing in the field or when the *value* attribute is set.

### max-length

An integer that specifies the maximum number of characters allowed by the element. Characters typed or pasted beyond the maximum are ignored.

### max-rows

An integer that specifies the maximum number of rows of text that are allowed within a crowd-text-area. Normally the element expands to accommodate new rows. If this is set, after the number of rows exceeds it, content scrolls upward out of view and a scrollbar control appears.

### name

A string used to represent the element's data in the output.

### placeholder

A string presented to the user as placeholder text. It disappears after the user puts something in the input area.

### rows

An integer that specifies the height of the element in rows of text.

### value

A preset that becomes the default if the worker does not provide input. The preset appears in a text field.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** none

## Output

This element outputs the name as a property name and the element's text contents as the value. Carriage returns in the text are represented as \n.

### Example Sample output for this element

```
[  
  {  
    "textInput1": "This is the text; the text that\nmakes the crowd go wild."  
  }  
]
```

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-toast

A subtle notification that temporarily appears on the display. Only one crowd-toast is visible.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following is an example of a Liquid template that uses the <crowd-toast> element. Copy the following code and save it in a file with the extension .html. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>  
  
<crowd-form>  
  <p>Find the official website for: <strong>{{ task.input.company }}</strong></p>  
  <p>Do not give Yelp pages, LinkedIn pages, etc.</p>  
  <p>Include the http:// prefix from the website</p>  
  <crowd-input name="website" placeholder="http://example.com"></crowd-input>  
  
  <crowd-toast duration="10000" opened>  
    This is a message that you want users to see when opening the template. This message  
    will disappear in 10 seconds.  
  </crowd-toast>  
  
</crowd-form>
```

## Attributes

The following attributes are supported by this element.

## duration

A number that specifies the duration, in milliseconds, that the notification appears on the screen.

## text

The text to display in the notification.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** none

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## crowd-toggle-button

A button that acts as an ON/OFF switch, toggling a state.

See an interactive example of an HTML template that uses this Crowd HTML Element in [CodePen](#).

The following example shows different ways you can use to use the `<crowd-toggle-button>` HTML element. Copy the following code and save it in a file with the extension `.html`. Open the file in any browser to preview and interact with this template.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
  <!--Toggle button without value-->
  <crowd-toggle-button name="toggleButtonWithoutValue"></crowd-toggle-button>

  <!--Toggle button with value-->
  <crowd-toggle-button name="toggleButtonWithValue" value="someValue"></crowd-toggle-button>

  <!--Toggle button disabled-->
  <crowd-toggle-button name="toggleButtonDisabled" disabled></crowd-toggle-button>

  <!--Toggle button marked invalid-->
  <crowd-toggle-button name="toggleButtonInvalid" invalid></crowd-toggle-button>

  <!--Toggle button marked required-->
  <crowd-toggle-button name="toggleButtonRequired" required></crowd-toggle-button>
</crowd-form>
```

## Attributes

The following attributes are supported by this element.

### checked

A Boolean switch that, if present, displays the button switched to the ON position.

### disabled

A Boolean switch that, if present, displays the button as disabled and prevents toggling.

### invalid

When in an off position, a button using this attribute, will display in an alert color. The standard is red, but may be changed in CSS. When toggled on, the button will display in the same color as other buttons in the on position.

### name

A string that is used to identify the answer submitted by the worker. This value matches a key in the JSON object that specifies the answer.

### required

A Boolean switch that, if present, requires the worker to provide input.

### value

A value used in the output as the property name for the element's Boolean state. Defaults to "on" if not provided.

## Element Hierarchy

This element has the following parent and child elements.

- **Parent elements:** [crowd-form \(p. 504\)](#)
- **Child elements:** none

## Output

This element outputs the name as the name of an object, containing the value as a property name and the element's state as Boolean value for the property. If no value for the element is specified, the property name defaults to "on".

### Example Sample output for this element

```
[  
  {  
    "theToggler": {  
      "on": true  
    }  
  }  
]
```

## See Also

For more information, see the following.

- [Use Amazon SageMaker Ground Truth to Label Data \(p. 165\)](#)
- [Crowd HTML Elements Reference \(p. 482\)](#)

## Augmented AI Crowd HTML Elements

The following Crowd HTML Elements are only available for Amazon Augmented AI human workflow tasks.

## Topics

- [crowd-textract-analyze-document \(p. 554\)](#)
- [crowd-rekognition-detect-moderation-labels \(p. 557\)](#)

## crowd-textract-analyze-document

A widget to enable human review of a Amazon Textract document analysis result.

### Attributes

The following attributes are supported by this element.

#### header

This is the text that is displayed as the header.

#### src

This is a link to the image to be analyzed by the worker.

#### initialValue

This sets initial values for attributes found in the worker UI.

The following is an example of an initialValue input:

```
[  
  {  
    "blockType": "KEY_VALUE_SET",  
    "confidence": 38.43309020996094,  
    "geometry": {  
      "boundingBox": {  
        "width": 0.32613086700439453,  
        "weight": 0.0942094624042511,  
        "left": 0.4833833575248718,  
        "top": 0.5227988958358765  
      },  
      "polygon": [  
        {"X": 0.123, "Y": 0.345}, ...  
      ]  
    }  
    "id": "8c97b240-0969-4678-834a-646c95da9cf4",  
    "relationships": [  
      {  
        "type": "CHILD",  
        "ids": [  
          "7ee7b7da-ee1b-428d-a567-55a3e3affa56",  
          "4d6da730-ba43-467c-a9a5-c6137ba0c472"  
        ]  
      },  
      {  
        "type": "VALUE",  
        "ids": [  
          "6ee7b7da-ee1b-428d-a567-55a3e3affa54"  
        ]  
      }  
    ],  
    "entityTypes": [  
      "KEY"  
    ],  
    "text": "Foo bar"
```

```
        },
    ]
```

### blockTypes

This determines the kind of analysis the workers can do. Only `KEY_VALUE_SET` is currently supported.

### keys

This specifies new keys and the associated text value the worker can add. The input values for `keys` can include the following elements:

- `importantFormKey` accepts strings, and is used to specify a single key.
- `importantFormKeyAliases` can be used to specify aliases that are acceptable alternatives to the keys supplied. Use this element to identify alternative spellings or presentations of your keys. This parameter accepts a list of one or more strings.

The following is an example of an input for `keys`.

```
[  
  {  
    importantFormKey: 'Address',  
    importantFormKeyAliases: [  
      'address',  
      'Addr.',  
      'Add.',  
    ]  
  },  
  {  
    importantFormKey: 'Last name',  
    importantFormKeyAliases: ['Surname']  
  }  
]
```

### no-key-edit

This prevents the workers from editing the keys of annotations passed through `initialValue`. This prevents workers from editing the keys that have been detected on your documents. This is required.

### no-geometry-edit

This prevents workers from editing the polygons of annotations passed through `initialValue`. For example, this would prevent the worker from editing the bounding box around a given key. This is required.

## Element Hierarchy

This element has the following parent and child elements.

- Parent elements – `crowd-form`
- Child elements – [full-instructions \(p. 556\)](#), [short-instructions \(p. 556\)](#)

## Regions

The following regions are supported by this element. You can use custom HTML and CSS code within these regions to format your instructions to workers. For example, use the `short-instructions` section to provide good and bad examples of how to complete a task.

### full-instructions

General instructions about how to work with the widget.

### short-instructions

Important task-specific instructions that are displayed in a prominent place.

## Example of a Worker Template Using the crowd Element

An example of a worker template using this crowd element would look like the following.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
{%
  capture s3_uri %}http://s3.amazonaws.com/
  {{ task.input.aiServiceRequest.document.s3Object.bucket }}/
  {{ task.input.aiServiceRequest.document.s3Object.name }}{% endcapture %}

<crowd-form>
  <crowd-textract-analyze-document
    src="{{ s3_uri | grant_read_access }}"
    initial-value="{{ task.input.selectedAiServiceResponse.blocks }}"
    header="Review the key-value pairs listed on the right and correct them if they don't
    match the following document."
    no-key-edit
    no-geometry-edit
    keys="{{ task.input.humanLoopContext.importantFormKeys }}"
    block-types="['KEY_VALUE_SET']"
  >
    <short-instructions header="Instructions">
      <style>
        .instructions {
          white-space: pre-wrap;
        }
        .instructionsImage {
          display: inline-block;
          max-width: 100%;
        }
      </style>
      <p class='instructions'>Click on a key-value block to highlight the corresponding
      key-value pair in the document.

      If it is a valid key-value pair, review the content for the value. If the content is
      incorrect, correct it.

      The text of the value is incorrect, correct it.
      

      A wrong value is identified, correct it.
      

      If it is not a valid key-value relationship, choose No.
      

      If you can't find the key in the document, choose Key not found.
      

      If the content of a field is empty, choose Value is blank.
      
    </short-instructions>
  </crowd-textract-analyze-document>
</crowd-form>
```

```

<b>Examples</b>
Key and value are often displayed next or below to each other.

Key and value displayed in one line.


Key and value displayed in two lines.


If the content of the value has multiple lines, enter all the text without line break.
Include all value text even if it extends beyond the highlight box.
</p>
</short-instructions>

<full-instructions header="Instructions"></full-instructions>
</crowd-extract-analyze-document>
</crowd-form>

```

## Output

The following is a sample of the output from this element. You can find a detailed explanation of this output in the Amazon Textract [AnalyzeDocument](#) API documentation.

```
{
    "AWS/Textract/AnalyzeDocument/Forms/V1": {
        "blocks": [
            {
                "blockType": "KEY_VALUE_SET",
                "id": "8c97b240-0969-4678-834a-646c95da9cf4",
                "relationships": [
                    {
                        "type": "CHILD",
                        "ids": ["7ee7b7da-ee1b-428d-a567-55a3e3affa56", "4d6da730-ba43-467c-a9a5-c6137ba0c472"]
                    },
                    {
                        "type": "VALUE",
                        "ids": ["6ee7b7da-ee1b-428d-a567-55a3e3affa54"]
                    }
                ],
                "entityTypes": ["KEY"],
                "text": "Foo bar baz"
            }
        ]
    }
}
```

## [crowd-rekognition-detect-moderation-labels](#)

A widget to enable human review of an Amazon Rekognition image moderation result.

### Attributes

The following attributes are supported by this element.

#### [header](#)

This is the text that is displayed as the header.

### src

This is a link to the image to be analyzed by the worker.

### categories

This supports `categories` as an array of strings **or** an array of objects where each object has a `name` field.

If the categories come in as objects, the following applies:

- The displayed categories are the value of the `name` field.
- The returned answer contains the **full** objects of any selected categories.

If the categories come in as strings, the following applies:

- The returned answer is an array of all the strings that were selected.

### exclusion-category

By setting this attribute you create a button underneath the categories in the UI.

- When a user chooses the button, all categories are deselected and disabled.
- Choosing the button again re-enables the categories so that users can choose them.
- If you submit after choosing the button, it returns an empty array.

## Element Hierarchy

This element has the following parent and child elements.

- Parent elements – crowd-form
- Child elements – [full-instructions \(p. 558\)](#), [short-instructions \(p. 558\)](#)

## Amazon Regions

The following Amazon Regions are supported by this element. You can use custom HTML and CSS code within these Regions to format your instructions to workers. For example, use the `short-instructions` section to provide good and bad examples of how to complete a task.

### full-instructions

General instructions about how to work with the widget.

### short-instructions

Important task-specific instructions that are displayed in a prominent place.

## Example Worker Template with the crowd Element

An example of a worker template using the `crowd` element would look like the following.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
{% capture s3_uri %}http://s3.amazonaws.com/
{{ task.input.aiServiceRequest.image.s3Object.bucket }}/
{{ task.input.aiServiceRequest.image.s3Object.name }}{% endcapture %}
```

```

<crowd-form>
  <crowd-rekognition-detect-moderation-labels
    categories='[
      {% for label in task.input.selectedAiServiceResponse.moderationLabels %}
      {
        name: "{{ label.name }}",
        parentName: "{{ label.parentName }}",
      },
      {% endfor %}
    ]'
    src="{{ s3_uri | grant_read_access }}"
    header="Review the image and choose all applicable categories."
  >
  <short-instructions header="Instructions">
    <style>
      .instructions {
        white-space: pre-wrap;
      }
    </style>
    <p class='instructions'>Review the image and choose all applicable categories.  
If no categories apply, choose None.
  </short-instructions>
  <b>Nudity</b>
  Visuals depicting nude male or female person or persons

  <b>Graphic Male Nudity</b>
  Visuals depicting full frontal male nudity, often close ups

  <b>Graphic Female Nudity</b>
  Visuals depicting full frontal female nudity, often close ups

  <b>Sexual Activity</b>
  Visuals depicting various types of explicit sexual activities and pornography

  <b>Illustrated Nudity or Sexual Activity</b>
  Visuals depicting animated or drawn sexual activity, nudity or pornography

  <b>Adult Toys</b>
  Visuals depicting adult toys, often in a marketing context

  <b>Female Swimwear or Underwear</b>
  Visuals depicting female person wearing only swimwear or underwear

  <b>Male Swimwear Or Underwear</b>
  Visuals depicting male person wearing only swimwear or underwear

  <b>Partial Nudity</b>
  Visuals depicting covered up nudity, for example using hands or pose

  <b>Revealing Clothes</b>
  Visuals depicting revealing clothes and poses, such as deep cut dresses

  <b>Graphic Violence or Gore</b>
  Visuals depicting prominent blood or bloody injuries

  <b>Physical Violence</b>
  Visuals depicting violent physical assault, such as kicking or punching

  <b>Weapon Violence</b>
  Visuals depicting violence using weapons like firearms or blades, such as shooting

  <b>Weapons</b>
  Visuals depicting weapons like firearms and blades

  <b>Self Injury</b>

```

```
Visuals depicting self-inflicted cutting on the body, typically in distinctive patterns
using sharp objects

<b>Emaciated Bodies</b>
Visuals depicting extremely malnourished human bodies

<b>Corpses</b>
Visuals depicting human dead bodies

<b>Hanging</b>
Visuals depicting death by hanging</p>
</short-instructions>

<full-instructions header="Instructions"></full-instructions>
</crowd-rekognition-detect-moderation-labels>
</crowd-form>
```

## Output

The following is a sample of the output from this element. For details about this output, see Amazon Rekognition [DetectModerationLabels](#) API documentation.

```
{
    "AWS/Rekognition/DetectModerationLabels/Image/V3": {
        "ModerationLabels": [
            { name: 'Gore', parentName: 'Violence' },
            { name: 'Corpses', parentName: 'Violence' },
        ]
    }
}
```

# Prepare and Analyze Datasets

Import, prepare, transform, visualize and analyze data with Amazon SageMaker Data Wrangler. You can integrate Data Wrangler into your machine learning workflows to simplify and streamline data pre-processing and feature engineering using little to no coding. You can also add your own Python scripts and transformations to customize your data prep workflow.

Import data from Amazon S3, Amazon Redshift, Amazon Athena, and use Data Wrangler to create sophisticated machine learning data prep workflows with built-in and custom data transformations and analysis including feature target leakage and quick modeling.

After you have defined a data prep workflow, or *data flow*, you can integrate it with SageMaker Processing, SageMaker Pipelines, and SageMaker Feature Store, simplify the task of processing, sharing and storing ML training data. You can also export your data flow to a python script and create a custom ML data prep pipeline.

For more information, see [Prepare ML Data with Amazon SageMaker Data Wrangler \(p. 577\)](#).

## Topics

- [Detect Pretraining Data Bias \(p. 561\)](#)
- [Prepare ML Data with Amazon SageMaker Data Wrangler \(p. 577\)](#)

## Detect Pretraining Data Bias

Algorithmic bias, discrimination, fairness, and related topics have been studied across disciplines such as law, policy, and computer science. A computer system might be considered biased if it discriminates against certain individuals or groups of individuals. The machine learning models powering these applications learn from data and this data could reflect disparities or other inherent biases. For example, the training data may not have sufficient representation of various demographic groups or may contain biased labels. The machine learning models trained on datasets that exhibit these biases could end up learning them and then reproduce or even exacerbate those biases in their predictions. The field of machine learning provides an opportunity to address biases by detecting them and measuring them at each stage of the ML lifecycle. You can use Amazon SageMaker Clarify to determine whether data used for training models encodes any bias.

Bias can be measured before training and after training, and monitored against baselines after deploying models to endpoints for inference. Pretraining bias metrics are designed to detect and measure bias in the raw data before it is used to train a model. The metrics used are model-agnostic because they do not depend on any model outputs. However, there are different concepts of fairness that require distinct measures of bias. Amazon SageMaker Clarify provides bias metrics to quantify various fairness criteria.

For additional information about bias metrics, see [Fairness Measures for Machine Learning in Finance](#).

## Amazon SageMaker Clarify Terms for Bias and Fairness

SageMaker Clarify uses the following terminology to discuss bias and fairness.

### Feature

An individual measurable property or characteristic of a phenomenon being observed, contained in a column for tabular data.

## Label

Feature that is the target for training a machine learning model. Referred to as the *observed label* or *observed outcome*.

## Predicted label

The label as predicted by the model. Also referred to as the *predicted outcome*.

## Sample

An observed entity described by feature values and label value, contained in a row for tabular data.

## Dataset

A collection of samples.

## Bias

An imbalance in the training data or the prediction behavior of the model across different groups, such as age or income bracket. Biases can result from the data or algorithm used to train your model. For instance, if an ML model is trained primarily on data from middle-aged individuals, it may be less accurate when making predictions involving younger and older people.

## Bias metric

A function that returns numerical values indicating the level of a potential bias.

## Bias report

A collection of bias metrics for a given dataset, or a combination of a dataset and a model.

## Positive label values

Label values that are favorable to a demographic group observed in a sample. In other words, designates a sample as having a *positive result*.

## Negative label values

Label values that are unfavorable to a demographic group observed in a sample. In other words, designates a sample as having a *negative result*.

## Group variable

Categorical column of the dataset that is used to form subgroups for the measurement of Conditional Demographic Disparity (CDD). Required only for this metric with regards to Simpson's paradox.

## Facet

A column or feature that contains the attributes with respect to which bias is measured.

## Facet value

The feature values of attributes that bias might favor or disfavor.

## Predicted probability

The probability, as predicted by the model, of a sample having a positive or negative outcome.

# Sample Notebooks

Amazon SageMaker Clarify provides the following sample notebook for bias detection:

- [Explainability and bias detection with Amazon SageMaker Clarify](#) – Use SageMaker Clarify to create a processing job for the detecting bias and explaining model predictions with feature attributions.

This notebook has been verified to run in Amazon SageMaker Studio only. If you need instructions on how to open a notebook in Amazon SageMaker Studio, see [Create or Open an Amazon SageMaker Studio Notebook \(p. 79\)](#). If you're prompted to choose a kernel, choose **Python 3 (Data Science)**.

### Topics

- [Measure Pretraining Bias \(p. 563\)](#)
- [Generate Reports for Bias in Pretraining Data in SageMaker Studio \(p. 572\)](#)

## Measure Pretraining Bias

Measuring bias in ML models is a first step to mitigating bias. Each measure of bias corresponds to a different notion of fairness. Even considering simple concepts of fairness leads to many different measures applicable in various contexts. For example, consider fairness with respect to age, and, for simplicity, that middle-aged and rest of the age groups are the two relevant demographics, referred to as *facets*. In the case of an ML model for lending, we may want small business loans to be issued to equal numbers of both demographics. Or, when processing job applicants, we may want to see equal numbers of members of each demographic hired. However, this approach may assume that equal numbers of both age groups apply to these jobs, so we may want to condition on the number that apply. Further, we may want to consider not whether equal numbers apply, but whether we have equal numbers of qualified applicants. Or, we may consider fairness to be an equal acceptance rate of qualified applicants across both age demographics, or, an equal rejection rate of applicants, or both. You might use datasets with different proportions of data on the attributes of interest. This imbalance can conflate the bias measure you choose. The models might be more accurate in classifying one facet than in the other. Thus, you need to choose bias metrics that are conceptually appropriate for the application and the situation.

We use the following notation to discuss the bias metrics. The conceptual model described here is for binary classification, where events are labeled as having only two possible outcomes in their sample space, referred to as positive (with value 1) and negative (with value 0). This framework is usually extensible to multiclassification in a straightforward way or to cases involving continuous valued outcomes when needed. In the binary classification case, positive and negative labels are assigned to outcomes recorded in a raw dataset for a favored facet *a* and for a disfavored facet *d*. These labels *y* are referred to as *observed labels* to distinguish them from the *predicted labels* *y'* that are assigned by a machine learning model during the training or inferences stages of the ML lifecycle. These labels are used to define probability distributions  $P_a(y)$  and  $P_d(y)$  for their respective facet outcomes.

- **labels:**
  - *y* represents the *n* observed labels for event outcomes in a training dataset.
  - *y'* represents the predicted labels for the *n* observed labels in the dataset by a trained model.
- **outcomes:**
  - A positive outcome (with value 1) for a sample, such as an application acceptance.
    - $n^{(1)}$  is the number of observed labels for positive outcomes (acceptances).
    - $n'^{(1)}$  is the number of predicted labels for positive outcomes (acceptances).
  - A negative outcome (with value 0) for a sample, such as an application rejection.
    - $n^{(0)}$  is the number of observed labels for negative outcomes (rejections).
    - $n'^{(0)}$  is the number of predicted labels for negative outcomes (rejections).
- **facet values:**
  - facet *a* – The feature value that defines a demographic that bias favors.
    - $n_a$  is the number of observed labels for the favored facet value:  $n_a = n_a^{(1)} + n_a^{(0)}$  the sum of the positive and negative observed labels for the value facet *a*.
    - $n'_a$  is the number of predicted labels for the favored facet value:  $n'_a = n'^{(1)}_a + n'^{(0)}_a$  the sum of the positive and negative predicted outcome labels for the facet value *a*. Note that  $n'_a = n_a$ .
  - facet *d* – The feature value that defines a demographic that bias disfavors.

- $n_d$  is the number of observed labels for the disfavored facet value:  $n_d = n_d^{(1)} + n_d^{(0)}$  the sum of the positive and negative observed labels for the facet value  $d$ .
- $n'_d$  is the number of predicted labels for the disfavored facet value:  $n'_d = n'_d^{(1)} + n'_d^{(0)}$  the sum of the positive and negative predicted labels for the facet value  $d$ . Note that  $n'_d = n_d$ .
- probability distributions for outcomes of the labeled facet data outcomes:
  - $P_a(y)$  is the probability distribution of the observed labels for facet  $a$ . For binary labeled data, this distribution is given by the ratio of the number of samples in facet  $a$  labeled with positive outcomes to the total number,  $P_a(y^1) = n_a^{(1)} / n_a$ , and the ratio of the number of samples with negative outcomes to the total number,  $P_a(y^0) = n_a^{(0)} / n_a$ .
  - $P_d(y)$  is the probability distribution of the observed labels for facet  $d$ . For binary labeled data, this distribution is given by the number of samples in facet  $d$  labeled with positive outcomes to the total number,  $P_d(y^1) = n_d^{(1)} / n_d$ , and the ratio of the number of samples with negative outcomes to the total number,  $P_d(y^0) = n_d^{(0)} / n_d$ .

Models trained on data biased by demographic disparities might learn and even exacerbate them. To identify bias in the data before expending resources to train models on it, SageMaker Clarify provides data bias metrics that you can compute on raw datasets before training. All of the pretraining metrics are model-agnostic because they do not depend on model outputs and so are valid for any model. The first bias metric examines facet imbalance, but not outcomes. It determines the extent to which the amount of training data is representative across different facets, as desired for the application. The remaining bias metrics compare the distribution of outcome labels in various ways for facets  $a$  and  $d$  in the data. The metrics that range over negative values can detect negative bias. The following table contains a cheat sheet for quick guidance and links to the pretraining bias metrics.

### Pretraining Bias Metrics

Bias metric	Description	Example question	Interpreting metric values
<a href="#">Class Imbalance (CI) (p. 567)</a>	Measures the imbalance in the number of members between different facet values.	Could there be age-based biases due to not having enough data for the demographic outside a middle-aged facet?	Normalized range: [-1, +1]  Interpretation: <ul style="list-style-type: none"> <li>• Positive values indicate the facet <math>a</math> has more training samples in the dataset.</li> <li>• Values near zero indicate the facets are balanced in the number of training samples in the dataset.</li> <li>• Negative values indicate the facet <math>d</math> has more training samples in the dataset.</li> </ul>
<a href="#">Difference in Proportions of Labels (DPL) (p. 568)</a>	Measures the imbalance of positive outcomes between different facet values.	Could there be age-based biases in ML predictions due to biased labeling of facet values in the data?	Range for normalized binary & multiclass facet labels: [-1,+1]

Bias metric	Description	Example question	Interpreting metric values
			<p>Range for continuous labels: <math>(-\infty, +\infty)</math></p> <p>Interpretation:</p> <ul style="list-style-type: none"> <li>Positive values indicate facet <math>a</math> has a higher proportion of positive outcomes.</li> <li>Values near zero indicate a more equal proportion of positive outcomes between facets.</li> <li>Negative values indicate facet <math>d</math> has a higher proportion of positive outcomes.</li> </ul>
<a href="#">Kullback-Leibler Divergence (KL) (p. 568)</a>	Measures how much the outcome distributions of different facets diverge from each other entropically.	How different are the distributions for loan application outcomes for different demographic groups?	<p>Range for binary, multiclass, continuous: <math>[0, +\infty)</math></p> <p>Interpretation:</p> <ul style="list-style-type: none"> <li>Values near zero indicate the labels are similarly distributed.</li> <li>Positive values indicate the label distributions diverge, the more positive the larger the divergence.</li> </ul>
<a href="#">Jensen-Shannon Divergence (JS) (p. 569)</a>	Measures how much the outcome distributions of different facets diverge from each other entropically.	How different are the distributions for loan application outcomes for different demographic groups?	<p>Range for binary, multiclass, continuous: <math>[0, +\infty)</math></p> <p>Interpretation:</p> <ul style="list-style-type: none"> <li>Values near zero indicate the labels are similarly distributed.</li> <li>Positive values indicate the label distributions diverge, the more positive the larger the divergence.</li> </ul>

Bias metric	Description	Example question	Interpreting metric values
<a href="#">L<sub>p</sub>-norm (LP) (p. 569)</a>	Measures a p-norm difference between distinct demographic distributions of the outcomes associated with different facets in a dataset.	How different are the distributions for loan application outcomes for different demographics?	<p>Range for binary, multicategory, continuous: <math>[0, +\infty)</math></p> <p>Interpretation:</p> <ul style="list-style-type: none"> <li>Values near zero indicate the labels are similarly distributed.</li> <li>Positive values indicate the label distributions diverge, the more positive the larger the divergence.</li> </ul>
<a href="#">Total Variation Distance (TVD) (p. 570)</a>	Measures half of the L <sub>1</sub> -norm difference between distinct demographic distributions of the outcomes associated with different facets in a dataset.	How different are the distributions for loan application outcomes for different demographics?	<p>Range for binary, multicategory, and continuous outcomes: <math>[0, +\infty)</math></p> <ul style="list-style-type: none"> <li>Values near zero indicates the labels are similarly distributed.</li> <li>Positive values indicates the label distributions diverge, the more positive the larger the divergence.</li> </ul>
<a href="#">Kolmogorov-Smirnov (KS) (p. 570)</a>	Measures maximum divergence between outcomes in distributions for different facets in a dataset.	Which college application outcomes manifest the greatest disparities by demographic group?	<p>Range of KS values for binary, multicategory, and continuous outcomes: <math>[0, +1]</math></p> <ul style="list-style-type: none"> <li>Values near zero indicate the labels were evenly distributed between facets in all outcome categories.</li> <li>Values near one indicate the labels for one category were all in one facet, so very imbalanced.</li> <li>Intermittent values indicate relative degrees of maximum label imbalance.</li> </ul>

Bias metric	Description	Example question	Interpreting metric values
<a href="#">Conditional Demographic Disparity (CDD) (p. 571)</a>	Measures the disparity of outcomes between different facets as a whole, but also by subgroups.	Do some groups have a larger proportion of rejections for college admission outcomes than their proportion of acceptances?	Range of CDD: [-1, +1] <ul style="list-style-type: none"> <li>Positive values indicate outcomes where facet <math>d</math> is rejected more than accepted.</li> <li>Near zero indicates no demographic disparity on average.</li> <li>Negative values indicate outcomes where facet <math>a</math> is rejected more than accepted.</li> </ul>

For additional information about bias metrics, see [Fairness Measures for Machine Learning in Finance](#).

### Topics

- [Class Imbalance \(CI\) \(p. 567\)](#)
- [Difference in Proportions of Labels \(DPL\) \(p. 568\)](#)
- [Kullback-Leibler Divergence \(KL\) \(p. 568\)](#)
- [Jensen-Shannon Divergence \(JS\) \(p. 569\)](#)
- [L<sub>p</sub>-norm \(LP\) \(p. 569\)](#)
- [Total Variation Distance \(TVD\) \(p. 570\)](#)
- [Kolmogorov-Smirnov \(KS\) \(p. 570\)](#)
- [Conditional Demographic Disparity \(CDD\) \(p. 571\)](#)

## Class Imbalance (CI)

Class imbalance (CI) bias occurs when a facet value  $d$  has fewer training samples when compared with another facet  $a$  in the dataset. This is because models preferentially fit the larger facets at the expense of the smaller facets and so can result in a higher training error for facet  $d$ . Models are also at higher risk of overfitting the smaller data sets, which can cause a larger test error for facet  $d$ . Consider the example where a machine learning model is trained primarily on data from middle-aged individuals, it might be less accurate when making predictions involving younger and older people.

The formula for the (normalized) facet imbalance measure:

$$CI = (n_a - n_d) / (n_a + n_d)$$

Where  $n_a$  is the number of members of facet  $a$  and  $n_d$  the number for facet  $d$ . Its values range over the interval [-1, 1].

- Positive CI values indicate the facet  $a$  has more training samples in the dataset and a value of 1 indicates the data only contains members of the facet  $a$ .
- Values of CI near zero indicate a more equal distribution of members between facets and a value of zero indicates a perfectly equal partition between facets and represents a balanced distribution of samples in the training data.

- Negative CI values indicate the facet  $d$  has more training samples in the dataset and a value of -1 indicates the data only contains members of the facet  $d$ .
- CI values near either of the extremes values of -1 or 1 are very imbalanced and are at a substantial risk of making biased predictions.

If a significant facet imbalance is found to exist among the facets, you might want to rebalance the sample before proceeding to train models on it.

## Difference in Proportions of Labels (DPL)

The difference in proportions of labels (DPL) compares the proportion of observed outcomes with positive labels for facet  $d$  with the proportion of observed outcomes with positive labels of facet  $a$  in a training dataset. For example, you could use it to compare the proportion of middle-aged individuals (facet  $a$ ) and other age groups (facet  $d$ ) approved for financial loans. Machine learning models try to mimic the training data decisions as closely as possible. So a machine learning model trained on a dataset with a high DPL is likely to reflect the same imbalance in its future predictions.

The formula for the difference in proportions of labels is as follows:

$$DPL = (q_a - q_d)$$

Where:

- $q_a = n_a^{(1)} / n_a$  is the proportion of facet  $a$  who have an observed label value of 1. For example, the proportion of a middle-aged demographic who get approved for loans. Here  $n_a^{(1)}$  represents the number of members of facet  $a$  who get a positive outcome and  $n_a$  the is number of members of facet  $a$ .
- $q_d = n_d^{(1)} / n_d$  is the proportion of facet  $d$  who have an observed label value of 1. For example, the proportion of people outside the middle-aged demographic who get approved for loans. Here  $n_d^{(1)}$  represents the number of members of the facet  $d$  who get a positive outcome and  $n_d$  the is number of members of the facet  $d$ .

If DPL is close enough to 0, then we say that *demographic parity* has been achieved.

For binary and multiclass facet labels, the DPL values range over the interval (-1, 1). For continuous labels, we set a threshold to collapse the labels to binary.

- Positive DPL values indicate that facet  $a$  has a higher proportion of positive outcomes when compared with facet  $d$ .
- Values of DPL near zero indicate a more equal proportion of positive outcomes between facets and a value of zero indicates perfect demographic parity.
- Negative DPL values indicate that facet  $d$  has a higher proportion of positive outcomes when compared with facet  $a$ .

Whether or not a high magnitude of DPL is problematic varies from one situation to another. In a problematic case, a high-magnitude DPL might be a signal of underlying issues in the data. For example, a dataset with high DPL might reflect historical biases or prejudices against age-based demographic groups that would be undesirable for a model to learn.

## Kullback-Leibler Divergence (KL)

The Kullback-Leibler divergence (KL) measures how much the observed label distribution of facet  $a$ ,  $P_a(y)$ , diverges from distribution of facet  $d$ ,  $P_d(y)$ . It is also known as the relative entropy of  $P_a(y)$  with respect to  $P_d(y)$  and quantifies the amount of information lost when moving from  $P_a(y)$  to  $P_d(y)$ .

The formula for the Kullback-Leibler divergence is as follows:

$$KL(P_a || P_d) = \sum_y P_a(y) \cdot \log[P_a(y)/P_d(y)]$$

It is the expectation of the logarithmic difference between the probabilities  $P_a(y)$  and  $P_d(y)$ , where the expectation is weighted by the probabilities  $P_a(y)$ . This is not a true distance between the distributions as it is asymmetric and does not satisfy the triangle inequality. The implementation uses natural logarithms, giving KL in units of nats. Using different logarithmic bases gives proportional results but in different units. For example, using base 2 gives KL in units of bits.

For example, assume that a group of applicants for loans have a 30% approval rate (facet *d*) and that the approval rate for other applicants (facet *a*) is 80%. The Kullback-Leibler formula gives you the label distribution divergence of facet *a* from facet *d* as follows:

$$KL = 0.8 * \ln(0.8/0.3) + 0.2 * \ln(0.2/0.7) = 0.53$$

There are two terms in the formula here because labels are binary in this example. This measure can be applied to multiple labels in addition to binary ones. For example, in a college admissions scenario, assume an applicant may be assigned one of three category labels:  $y_i = \{y_0, y_1, y_2\} = \{\text{rejected}, \text{waitlisted}, \text{accepted}\}$ .

Range of values for the KL metric for binary, multiclass, and continuous outcomes is  $[0, +\infty)$ .

- Values near zero mean the outcomes are similarly distributed for the different facets.
- Positive values mean the label distributions diverge, the more positive the larger the divergence.

## Jensen-Shannon Divergence (JS)

The Jensen-Shannon divergence (JS) measures how much the label distributions of different facets diverge from each other entropically. It is based on the Kullback-Leibler divergence, but it is symmetric.

The formula for the Jensen-Shannon divergence is as follows:

$$JS = \frac{1}{2} * [KL(P_a || P) + KL(P_d || P)]$$

Where  $P = \frac{1}{2}(P_a + P_d)$ , the average label distribution across facets *a* and *d*.

The range of JS values for binary, multiclass, continuous outcomes is  $[0, \ln(2)]$ .

- Values near zero mean the labels are similarly distributed.
- Positive values mean the label distributions diverge, the more positive the larger the divergence.

This metric indicates whether there is a big divergence in one of the labels across facets.

## $L_p$ -norm (LP)

The  $L_p$ -norm (LP) measures the  $p$ -norm distance between the facet distributions of the observed labels in a training dataset. This metric is non-negative and so cannot detect reverse bias.

The formula for the  $L_p$ -norm is as follows:

$$L_p(P_a, P_d) = (\sum_y \|P_a - P_d\|_p^p)^{1/p}$$

Where the  $p$ -norm distance between the points  $x$  and  $y$  is defined as follows:

$$L_p(x, y) = (|x_1 - y_1|^p + |x_2 - y_2|^p + \dots + |x_n - y_n|^p)^{1/p}$$

The 2-norm is the Euclidean norm. Assume you have an outcome distribution with three categories, for example,  $y_i = \{y_0, y_1, y_2\} = \{\text{accepted}, \text{waitlisted}, \text{rejected}\}$  in a college admissions multicategory scenario. You take the sum of the squares of the differences between the outcome counts for facets  $a$  and  $d$ . The resulting Euclidean distance is calculated as follows:

$$L_2(P_a, P_d) = [(n_a^{(0)} - n_d^{(0)})^2 + (n_a^{(1)} - n_d^{(1)})^2 + (n_a^{(2)} - n_d^{(2)})^2]^{1/2}$$

Where:

- $n_a^{(i)}$  is number of the  $i$ th category outcomes in facet  $a$ : for example  $n_a^{(0)}$  is number of facet  $a$  acceptances.
- $n_d^{(i)}$  is number of the  $i$ th category outcomes in facet  $d$ : for example  $n_d^{(2)}$  is number of facet  $d$  rejections.

The range of LP values for binary, multicategory, and continuous outcomes is  $[0, \sqrt{2}]$ , where:

- Values near zero mean the labels are similarly distributed.
- Positive values mean the label distributions diverge, the more positive the larger the divergence.

## Total Variation Distance (TVD)

The total variation distance data bias metric (TVD) is half the  $L_1$ -norm. The TVD is the largest possible difference between the probability distributions for label outcomes of facets  $a$  and  $d$ . The  $L_1$ -norm is the Hamming distance, a metric used compare two binary data strings by determining the minimum number of substitutions required to change one string into another. If the strings were to be copies of each other, it determines the number of errors that occurred when copying. In the bias detection context, TVD quantifies how many outcomes in facet  $a$  would have to be changed to match the outcomes in facet  $d$ .

The formula for the Total variation distance is as follows:

$$\text{TVD} = \frac{1}{2} \cdot L_1(P_a, P_d)$$

For example, assume you have an outcome distribution with three categories,  $y_i = \{y_0, y_1, y_2\} = \{\text{accepted}, \text{waitlisted}, \text{rejected}\}$ , in a college admissions multicategory scenario. You take the differences between the counts of facets  $a$  and  $d$  for each outcome to calculate TVD. The result is as follows:

$$L_1(P_a, P_d) = |n_a^{(0)} - n_d^{(0)}| + |n_a^{(1)} - n_d^{(1)}| + |n_a^{(2)} - n_d^{(2)}|$$

Where:

- $n_a^{(i)}$  is number of the  $i$ th category outcomes in facet  $a$ : for example  $n_a^{(0)}$  is number of facet  $a$  acceptances.
- $n_d^{(i)}$  is number of the  $i$ th category outcomes in facet  $d$ : for example  $n_d^{(2)}$  is number of facet  $d$  rejections.

The range of TVD values for binary, multicategory, and continuous outcomes is  $[0, 1]$ , where:

- Values near zero mean the labels are similarly distributed.
- Positive values mean the label distributions diverge, the more positive the larger the divergence.

## Kolmogorov-Smirnov (KS)

The Kolmogorov-Smirnov bias metric (KS) is equal to the maximum divergence between labels in the distributions for facets  $a$  and  $d$  of a dataset. The two-sample KS test implemented by SageMaker Clarify complements the other measures of label imbalance by finding the most imbalanced label.

The formula for the Kolmogorov-Smirnov metric is as follows:

$$KS = \max(|P_a(y) - P_d(y)|)$$

For example, assume a group of applicants (facet *a*) to college are rejected, waitlisted, or accepted at 40%, 40%, 20% respectively and that these rates for other applicants (facet *d*) are 20%, 10%, 70%. Then the Kolmogorov-Smirnov bias metric value is as follows:

$$KS = \max(|0.4-0.2|, |0.4-0.3|, |0.2-0.7|) = 0.5$$

This tells us the maximum divergence between facet distributions is 0.5 and occurs in the acceptance rates. There are three terms in the equation because labels are multiclass of cardinality three.

The range of LP values for binary, multiclass, and continuous outcomes is [0, +1], where:

- Values near zero indicate the labels were evenly distributed between facets in all outcome categories. For example, both facets applying for a loan got 50% of the acceptances and 50% of the rejections.
- Values near one indicate the labels for one outcome were all in one facet. For example, facet *a* got 100% of the acceptances and facet *d* got none.
- Intermittent values indicate relative degrees of maximum label imbalance.

## Conditional Demographic Disparity (CDD)

The demographic disparity metric (DD) determines whether facet *d* has a larger proportion of the rejected outcomes in the dataset than of the accepted outcomes. For example, in the case of college admissions, if women applicants comprised 60% of the rejected applicants and comprised only 50% of the accepted applicants, we say that there is *demographic disparity* because the rate at which women were rejected exceeds the rate at which they are accepted.

The formula for the demographic disparity for the less favored facet *d* is as follows:

$$DD_d = n_d^{(0)} / n^{(0)} - n_d^{(1)} / n^{(1)} = P_d^R(y^0) - P_d^A(y^1)$$

Where:

- $n^{(0)} = n_a^{(0)} + n_d^{(0)}$  is the number of rejected outcomes in the dataset.
- $n^{(1)} = n_a^{(1)} + n_d^{(1)}$  is the number of accepted outcomes in the dataset.
- $P_d^R(y^0)$  is the proportion of rejected outcomes (with value 0) in facet *d*.
- $P_d^A(y^1)$  is the proportion of accepted outcomes (value 1) in facet *d*.

For the college admission example, the demographic disparity is  $DD = 0.6 - 0.5 = 0.1$ .

A conditional demographic disparity (CDD) metric that conditions DD on attributes that define a strata of subgroups on the dataset is needed to rule out Simpson's paradox. The regrouping can provide insights into the cause of apparent demographic disparities for less favored facets. The classic case arose in the case of Berkeley admissions where men were accepted at a higher rate overall than women. However, when departmental subgroups were examined, women were shown to have higher admission rates than men by department. The explanation was that women had applied to departments with lower acceptance rates than men had. Examining the subgrouped acceptance rates revealed that women were actually accepted at a higher rate than men for the departments with lower acceptance rates.

The CDD metric gives a single measure for all of the disparities found in the subgroups defined by an attribute of a dataset by averaging them. It is defined as the weighted average of demographic disparities ( $DD_i$ ) for each of the subgroups, with each subgroup disparity weighted in proportion to the number of observations it contains. The formula for the conditional demographic disparity is as follows:

$$CDD = (1/n) \cdot \sum_i n_i \cdot DD_i$$

Where:

- $\sum_i n_i = n$  is the total number of observations and  $n_i$  is the number of observations for each subgroup.
- $DD_i = n_i^{(0)}/n^{(0)} - n_i^{(1)}/n^{(1)} = P_i^R(y^0) - P_i^A(y^1)$  is the demographic disparity for the  $i$ th subgroup.

The demographic disparity for a subgroup ( $DD_i$ ) are the difference between the proportion of rejected outcomes and the proportion of accepted outcomes for each subgroup.

The range of DD values for binary outcomes is (-1, +1).

- +1: when there are no rejections in facet  $a$  or subgroup and no acceptances in facet  $d$  or subgroup
- Positive values indicate there is a demographic disparity as facet  $d$  or subgroup has a smaller proportion of the rejected outcomes in the dataset than of the accepted outcomes. The higher the value the greater the disparity.
- Negative values indicate there is a demographic disparity as facet  $a$  or subgroup has a larger proportion of the rejected outcomes in the dataset than of the accepted outcomes. The lower the value the greater the disparity.
- -1: when there are no rejections in facet  $d$  or subgroup and no acceptances in facet  $a$  or subgroup

If you don't condition on anything then CDD is zero if and only if DPL is zero.

This metric is useful for exploring the concepts of direct and indirect discrimination and of objective justification in EU and UK non-discrimination law and jurisprudence. For additional information, see [Why Fairness Cannot Be Automated](#).

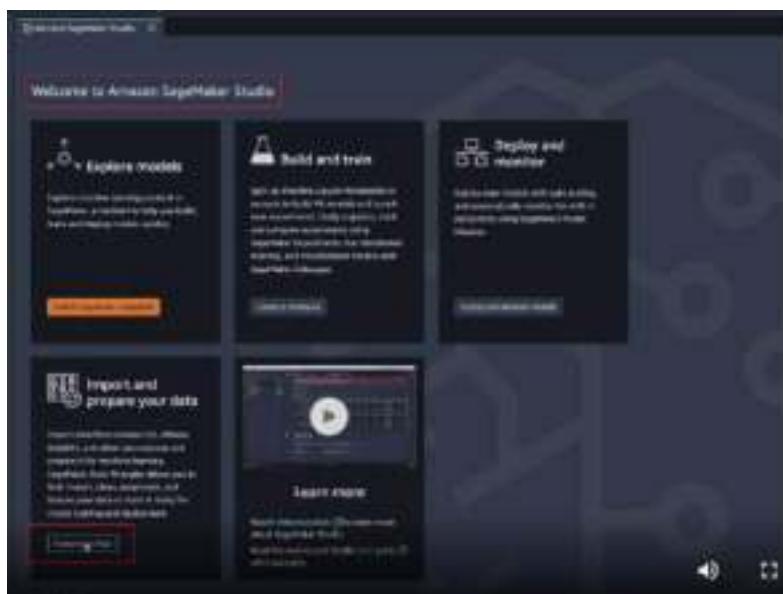
## Generate Reports for Bias in Pretraining Data in SageMaker Studio

SageMaker Clarify is integrated with Amazon SageMaker Data Wrangler, which can help you identify bias during data preparation without having to write your own code. Data Wrangler provides an end-to-end solution to import, prepare, transform, featurize, and analyze data with Amazon SageMaker Studio. For an overview of the Data Wrangler data prep workflow, see [Prepare ML Data with Amazon SageMaker Data Wrangler \(p. 577\)](#). You specify attributes of interest, such as gender or age, and SageMaker Clarify runs a set of algorithms to detect the presence of bias in those attributes. After the algorithm runs, SageMaker Clarify provides a visual report with a description of the sources and severity of possible bias so that you can plan steps to mitigate. For example, in a financial dataset that contains few examples of business loans to one age group as compared to others, SageMaker flags the imbalance so that you can avoid a model that disfavors that age group.

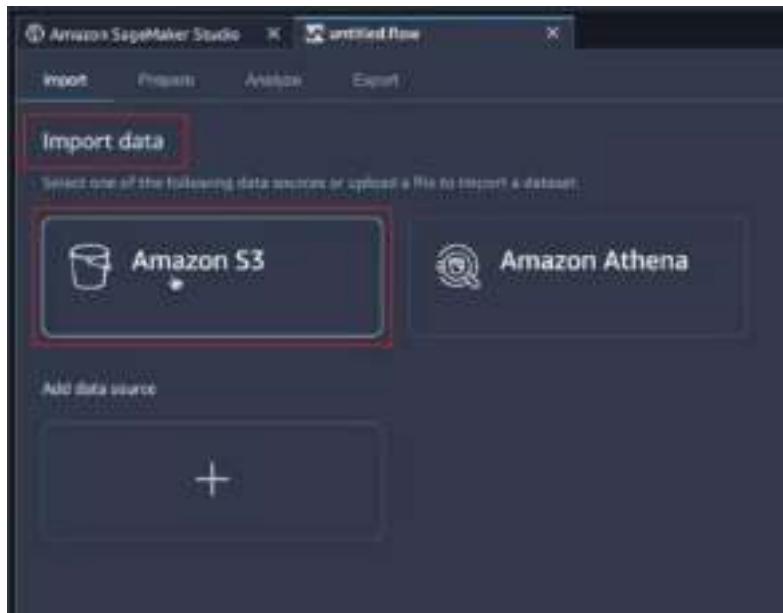
### To analyze and report on data bias

To get started with Data Wrangler, see [Get Started with Data Wrangler \(p. 578\)](#).

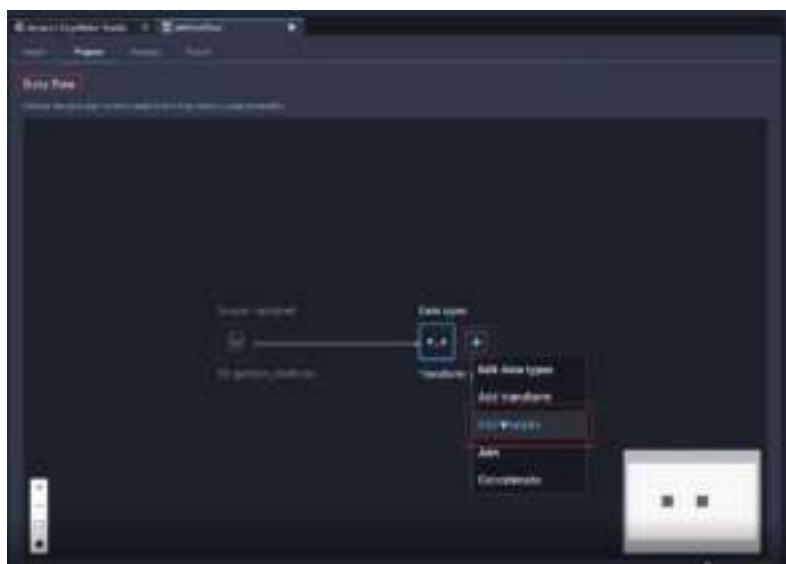
1. Open Amazon SageMaker Studio and choose **Create Data Flow** from the **Import and prepare your data** tile.



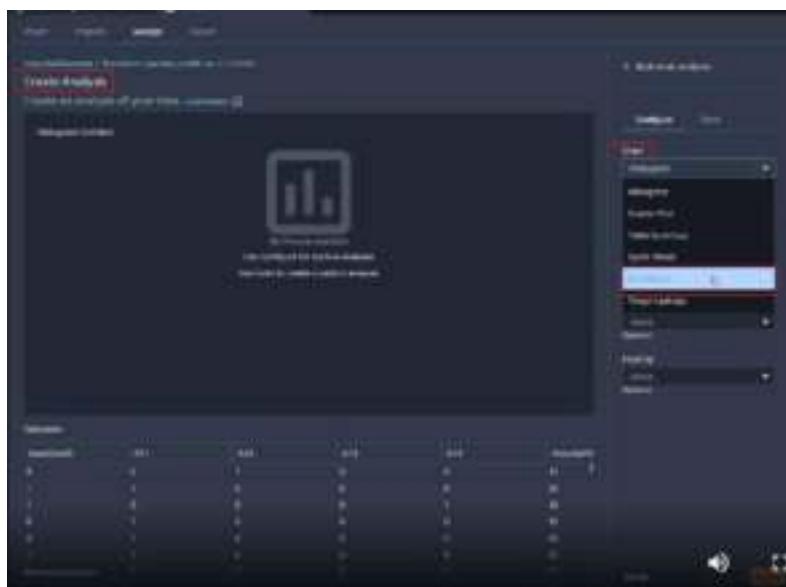
- From the **Import data** tab, choose **Amazon S3** and then specify your data source on the **Data sources/S3 source** page.



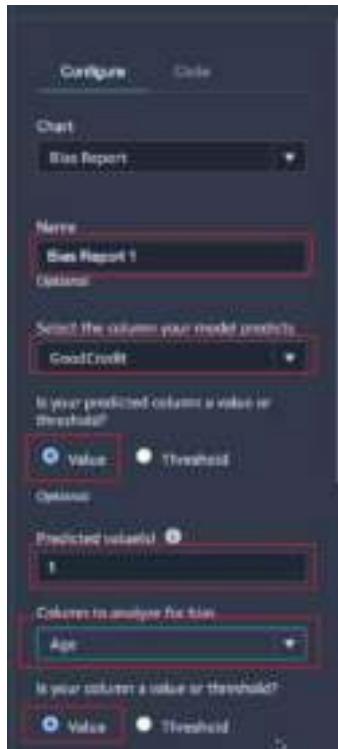
- After you have imported your data, choose the plus sign on the **Data flow** page and then choose **Add Analysis**.



4. On the **Create Analysis** page, go to the **Configure** panel and then choose **Bias Report** from the **Chart** menu.



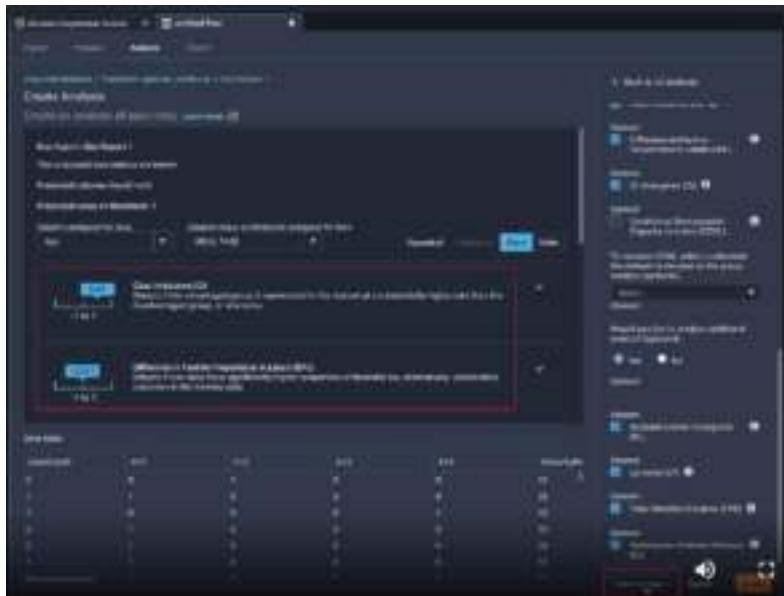
5. Configure the bias report by providing the **Name**, the column to predict and whether it is a value or threshold, the column to analyze for bias (the facet) and whether it is a value or threshold.



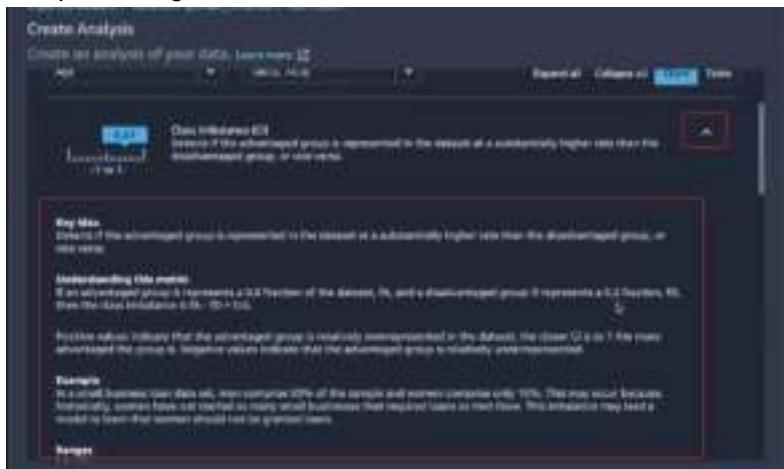
6. Continue configuring the bias report by choosing the bias metrics.



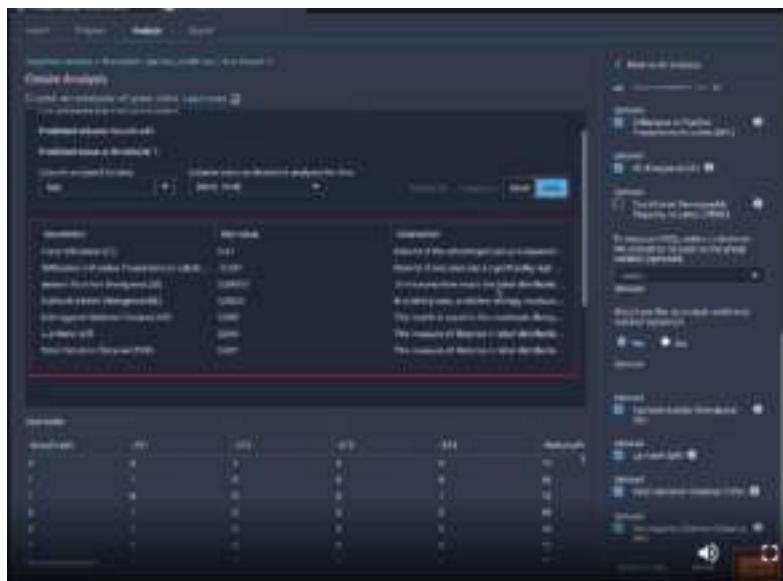
7. Choose **Check for bias** to generate and view the bias report. Scroll down to view all of the reports.



8. Choose the carrot to the right of the bias metric description to see documentation that can help you interpret the significance of the metric values.



9. To view a table summary of the bias metric values, choose the table. You can save the report for export by choosing **Create** in the lower-right corner of the page.



10. On the page where your data bias reports are stored, choose the **Export** tab to download the reports.



## Prepare ML Data with Amazon SageMaker Data Wrangler

Amazon SageMaker Data Wrangler (Data Wrangler) is a feature of SageMaker Studio that provides an end-to-end solution to import, prepare, transform, featurize, and analyze data. You can integrate a Data Wrangler data flow into your machine learning (ML) workflows to simplify and streamline data pre-processing and feature engineering using little to no coding. You can also add your own Python scripts and transformations to customize a For high-level security needs Data Wrangler data prep workflow.

The following are the core functionalities that Data Wrangler provides to help you analyze and prepare data for machine learning applications.

- **Import** – Connect to and import data from Amazon Simple Storage Service (Amazon S3), Amazon Athena (Athena), and Amazon Redshift.
- **Data Flow** – Create a data flow to define a series of ML data prep steps. You can use a flow to combine datasets from different data sources, identify the number and types of transformations you want to apply to datasets, and define a data prep workflow that can be easily integrated into an ML pipeline.
- **Transform** – Clean and transform your dataset using standard *transforms* like string, vector, and numeric data formatting tools. Featurize your data using transforms like text and date/time embedding and categorical encoding.
- **Analyze** – Analyze features in your dataset at any point in your flow. Data Wrangler includes built-in data visualization tools like scatter plots and histograms, as well as data analysis tools like target leakage analysis and quick modeling to understand feature correlation.
- **Export** – Data Wrangler offers export options to other SageMaker services, including Data Wrangler jobs, Feature Store, and pipelines, making it easy to integrate your data prep flow into your ML workflow. You can also export your Data Wrangler flow to Python code.

To start using Data Wrangler, see [Get Started with Data Wrangler \(p. 578\)](#).

### Topics

- [Get Started with Data Wrangler \(p. 578\)](#)
- [Import \(p. 588\)](#)
- [Create and Use a Data Wrangler Flow \(p. 618\)](#)
- [Transform Data \(p. 621\)](#)
- [Analyze and Visualize \(p. 638\)](#)
- [Export \(p. 645\)](#)
- [Shut Down Data Wrangler \(p. 650\)](#)
- [Update Data Wrangler \(p. 581\)](#)
- [Security and Permissions \(p. 653\)](#)
- [Release Notes \(p. 661\)](#)
- [Troubleshoot \(p. 662\)](#)

## Get Started with Data Wrangler

Amazon SageMaker Data Wrangler is a feature in SageMaker Studio. Use this section to learn how to access and get started using Data Wrangler. Do the following:

1. Complete each step in [Prerequisites \(p. 578\)](#).
2. Follow the procedure in [Access Data Wrangler \(p. 579\)](#) to start using Data Wrangler.

## Prerequisites

To use Data Wrangler, you must do the following:

1. To use Data Wrangler, you need access to a **m5.4xlarge** Amazon Elastic Compute Cloud (Amazon EC2) instance. To learn how to view your quotas, and if necessary, request a quota increase, see [Amazon service quotas](#).
2. Configure the required permissions described in [Security and Permissions \(p. 653\)](#).

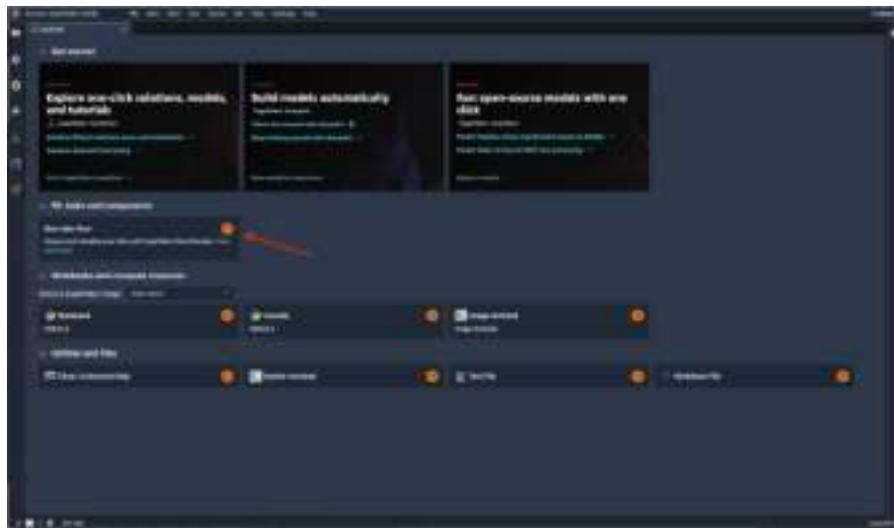
To use Data Wrangler, you need an active SageMaker Studio instance. To learn how to launch a new instance, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#). When your Studio instance is **Ready**, use the instructions in [Access Data Wrangler \(p. 579\)](#).

## Access Data Wrangler

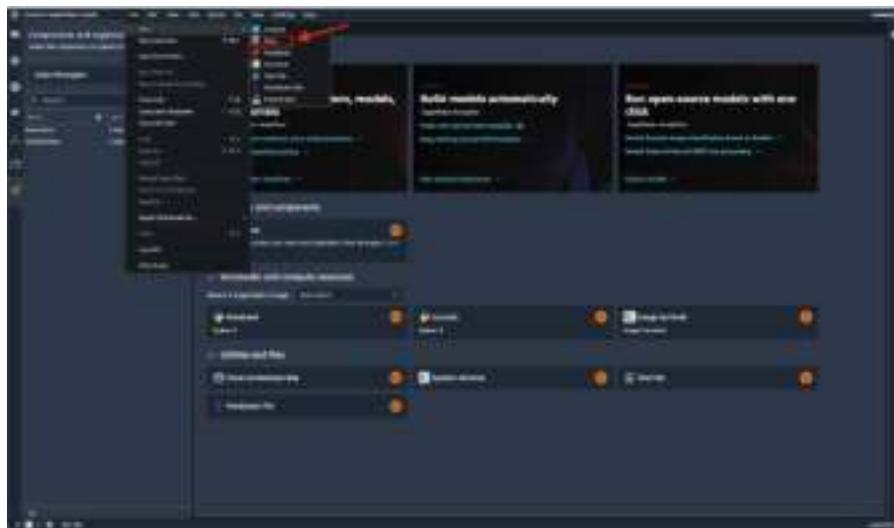
The following procedure assumes you have completed the [Prerequisites \(p. 578\)](#).

### To access Data Wrangler in Studio:

1. Next to the user you want to use to launch Studio, select **Open Studio**.
2. When Studio opens, select the + sign on the **New data flow** card under **ML tasks and components**. This creates a new directory in Studio with a .flow file inside, which contains your data flow. The .flow file automatically opens in Studio.



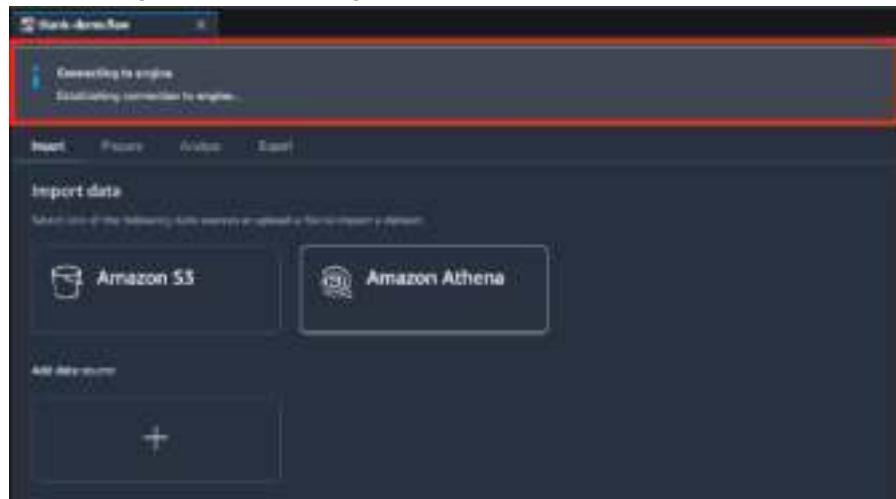
You can also create a new flow by selecting **File**, then **New**, and choosing **Flow** in the top navigation bar.



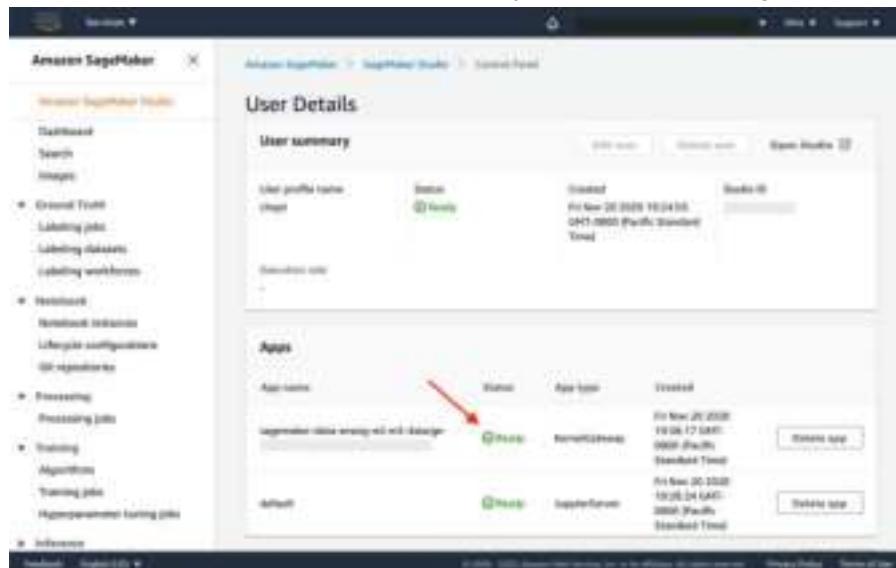
3. (Optional) Rename the new directory and the .flow file.
4. When you create a new .flow file in Studio, you may see a message at the top of the Data Wrangler interface that says:

## Connecting to engine

Establishing connection to engine...



This message persists as long as the **KernelGateway** app on your **User Details** page is **Pending**. To see the status of this app, in the SageMaker console on the **Amazon SageMaker Studio** page, select the name of the user you are using to access Studio. On the **User Details** page, you see a **KernelGateway** app under **Apps**. Wait until this app status is **Ready** to start using Data Wrangler. This can take around 5 minutes the first time you launch Data Wrangler.



5. To get started, choose a data source and use it to import a dataset. See [Import \(p. 588\)](#) to learn more.

When you import a dataset, it appears in your data flow. To learn more, see [Create and Use a Data Wrangler Flow \(p. 618\)](#).

6. After you import a dataset, Data Wrangler automatically infers the type of data in each column. Choose + next to the **Data types** step and select **Edit data types**.

### Important

After you add transforms to the **Data types** step, you cannot bulk-update column types using **Update types**.

7. Use the data flow to add transforms and analyses. To learn more see [Transform Data \(p. 621\)](#) and [Analyze and Visualize \(p. 638\)](#).
8. To export a complete data flow, choose **Export** and choose an export option. To learn more, see [Export \(p. 645\)](#).
9. Finally, choose the **Components and registries** icon, and select **Data Wrangler** from the dropdown list to see all .flow files you've created. You can use this menu to find and move between data flows.



After you have launched Data Wrangler, you can use the following section to walk through how you might use Data Wrangler to create an ML data prep flow.

## Update Data Wrangler

It is recommended that you periodically update the Data Wrangler Studio app to access the latest features and updates. The data wrangler app name starts with **sagemaker-data-wrang**. To learn how to update a SageMaker Studio app, see [Update Studio Apps \(p. 118\)](#).

## Demo: Data Wrangler Titanic Dataset Walkthrough

The following sections provide a walkthrough to help you get started using Data Wrangler. This walkthrough assumes that you have already followed the steps in [Access Data Wrangler \(p. 579\)](#) and have a new data flow file open that you intend to use for the demo. You may want to rename this .flow file to something similar to titanic-demo.flow.

This walk through uses the [Titanic dataset](#). This data set contains the survival status, age, gender, and class (which serves as a proxy for economic status) of passengers aboard the maiden voyage of the RMS Titanic in 1912.

In this tutorial, you:

- Upload the [Titanic dataset](#) to Amazon Simple Storage Service (Amazon S3), and then import this dataset into Data Wrangler.
- Analyze this dataset using Data Wrangler analyses.
- Define a data flow using Data Wrangler data transforms.
- Export your flow to a Jupyter Notebook that you can use to create a Data Wrangler job.
- Process your data, and kick off a SageMaker training job to train a XGBoost Binary Classifier.

## Upload Dataset to S3 and Import

To get started, download the [Titanic dataset](#) and upload it to an S3 bucket in the Amazon Region in which you want to complete this demo.

If you are a new user of Amazon S3, you can do this using drag and drop in the Amazon S3 console. To learn how, see [Uploading Files and Folders by Using Drag and Drop](#) in the Amazon Simple Storage Service Console User Guide.

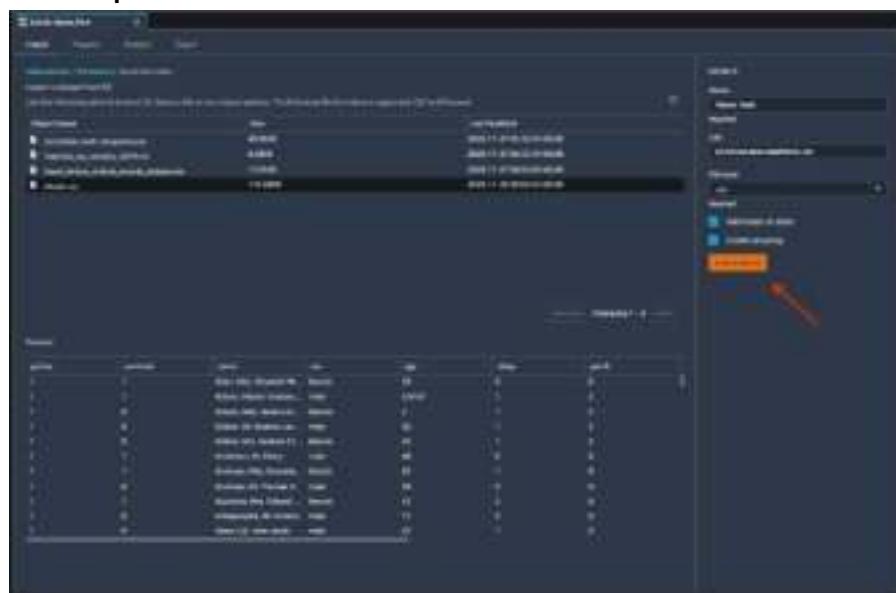
**Important**

Upload your dataset to an S3 bucket in the same Amazon Region you want to use to complete this demo.

When your dataset has been successfully uploaded to Amazon S3, it you can import it into Data Wrangler.

### Import the Titanic dataset to Data Wrangler

1. Select the **Import** tab in your Data Wrangler flow file.
2. Select **Amazon S3**.
3. Use the **Import a dataset from S3** table to find the bucket to which you added the Titanic dataset. Choose the Titanic dataset CSV file to open the **Details** pane.
4. Under **Details**, the **File type** should be CSV. Choose **Add header to table** to specify that the first row of the dataset is a header. You can also name the dataset something more friendly, such as *Titanic-train*.
5. Select **Import dataset**.



When your dataset is imported into Data Wrangler, it appears in your data flow. You can view your data flow at any time by selecting the **Prepare** tab. In the next section, you use this data flow to add analysis and transform steps.

## Data Flow

In the data flow section, you'll notice that the only steps in the data flow are your recently imported dataset and a **Data type** step. After applying transformations, you can come back to this tab see what the data flow looks like. Now, add some basic transformations under the **Prepare** and **Analyze** tabs.

## Prepare and Visualize

Data Wrangler has built-in transformations and visualizations that you can use to analyze, clean, and transform your data.

In the **Prepare** tab, all built-in transformations are listed in the right panel, which also contains an area in which you can add custom transformations. The following use case showcases how to use these transformations.

## Data Exploration

First, create a table summary of the data using an analysis. Do the following:

1. Choose the + next to the **Data type** step in your data flow and select **Add analysis**.
2. In the **Analysis** area, select **Table summary** from the dropdown list.
3. Give the table summary a **Name**.
4. Select **Preview** to preview the table that will be created.
5. Choose **Create** to save it to your data flow. It appears under **All Analyses**.

Using the statistics you see, you can make observations similar to the following about this dataset:

- Fare average (mean) is around \$33, while the max is over \$500. This column likely has outliers.
- This dataset uses ? to indicate missing values. A number of columns have missing values: *cabin*, *embarked*, and *home.dest*
- The age category is missing over 250 values.

Choose **Prepare** to go back to the data flow. Next, clean your data using the insights gained from these stats.

## Drop Unused Columns

Using the analysis from the previous section, clean up the dataset to prepare it for training. To add a new transform to your data flow, choose + next to the **Data type** step in your data flow and choose **Add transform**.

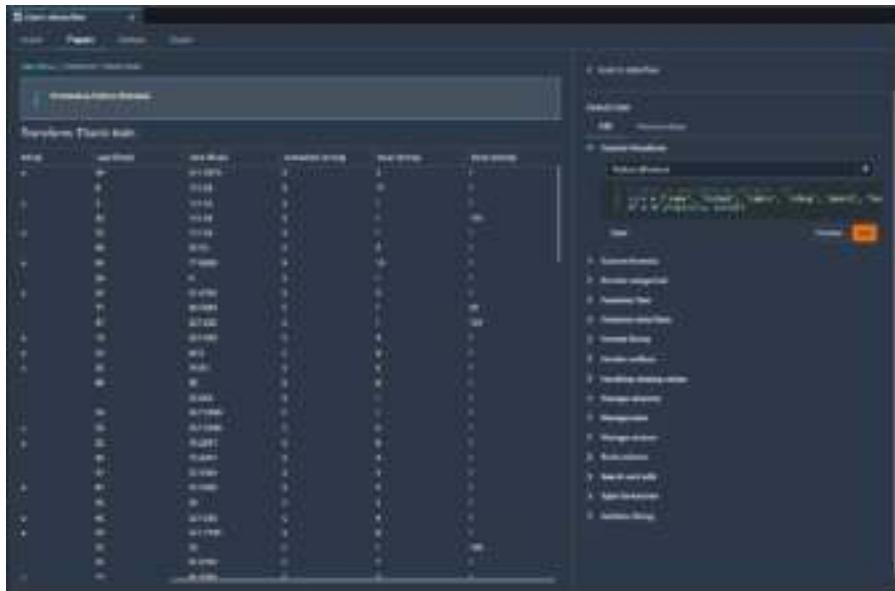
First, drop columns that you don't want to use for training. You can use [Pandas](#) data analysis library to do this, or you can use one of the built-in transforms.

To do this using Pandas, do the following:

1. In the **Custom Transform** section, select **Python (Pandas)** from the dropdown list.
2. Enter the following in the code box.

```
cols = ['name', 'ticket', 'cabin', 'sibsp', 'parch', 'home.dest','boat', 'body']
df = df.drop(cols, axis=1)
```

3. Choose **Preview** to preview the change and then choose **Add** to add the transformation.



To use the built-in transformations, do the following:

1. Choose **Manage columns** from the right panel.
  2. For **Input column**, choose **cabin**, and choose **Preview**.
  3. Verify that the **cabin** column has been dropped, then choose **Add**.
  4. Repeat these steps for the following columns: **ticket**, **name**, **sibsp**, **parch**, **home.dest**, **boat**, and **body**.

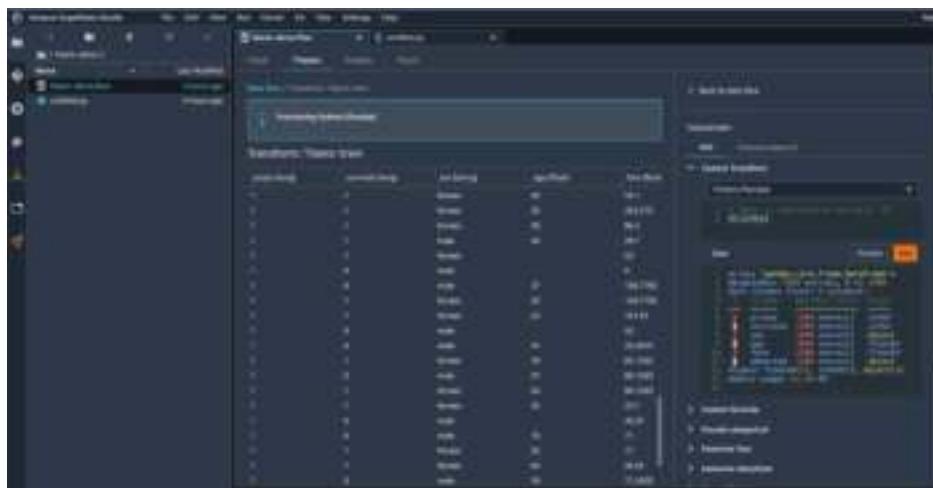
## Clean up Missing Values

Now, clean up missing values. You can do this with the **Handling missing values** transform group.

A number of columns have missing values. Of the remaining columns, *age* and *fare* contain missing values. Inspect this using the **Custom Transform**.

Using the **Python (Pandas)** option, use the following to quickly review the number of entries in each column:

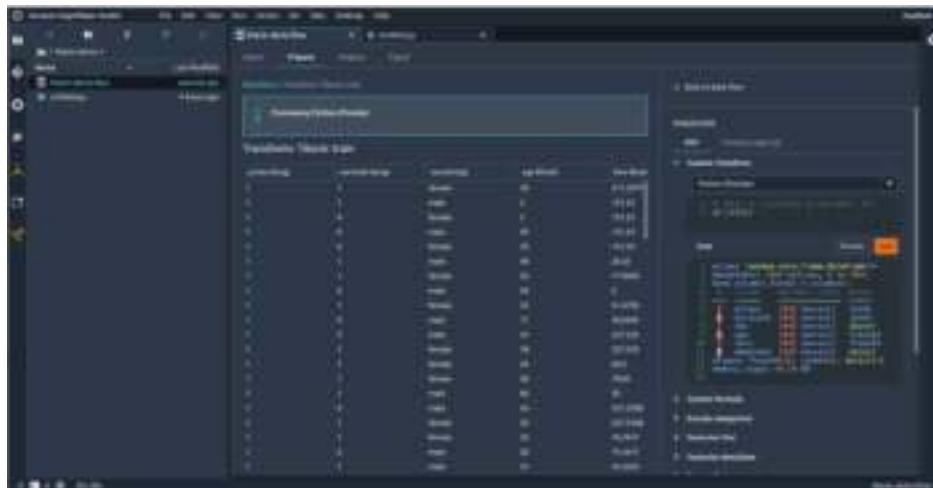
```
df.info()
```



To drop rows with missing values in the *age* category, do the following:

1. Choose **Handling missing values**.
2. Choose **Drop missing** for the **Transformer**.
3. Choose **Drop Rows** for the **Dimension**.
4. Choose *age* for the **Input column**.
5. Choose **Preview** to see the new data frame, and then choose **Add** to add the transform to your flow.
6. Repeat the same process for *fare*.

You can use `df.info()` in the **Custom transform** section to confirm that all rows now have 1,045 values.



### Custom Pandas: Encode

Try flat encoding using Pandas. Encoding categorical data is the process of creating a numerical representation for categories. For example, if your categories are Dog and Cat, you may encode this information into two vectors: [1, 0] to represent Dog, and [0, 1] to represent Cat.

1. In the **Custom Transform** section, choose **Python (Pandas)** from the dropdown list.
2. Enter the following in the code box.

```
import pandas as pd

dummies = []
cols = ['pclass', 'sex', 'embarked']
for col in cols:
    dummies.append(pd.get_dummies(df[col]))

encoded = pd.concat(dummies, axis=1)

df = pd.concat((df, encoded), axis=1)
```

3. Choose **Preview** to preview the change. The encoded version of each column is added to the dataset.
4. Choose **Add** to add the transformation.

### Custom SQL: SELECT Columns

Now, select the columns you want to keep using SQL. For this demo, select the columns listed in the following **SELECT** statement. Because *survived* is your target column for training, put that column first.

1. In the **Custom Transform** section, select **SQL (PySpark SQL)** from the dropdown list.
2. Enter the following in the code box.

```
SELECT survived, age, fare, 1, 2, 3, female, male, C, Q, S FROM df;
```

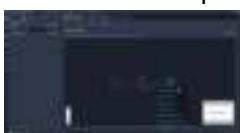
3. Choose **Preview** to preview the change. The columns listed in your **SELECT** statement above are the only remaining columns.
4. Choose **Add** to add the transformation.

## Export

When you've finished creating a data flow, you have a number of export options. The following section explains how to export to a Data Wrangler job notebook. A Data Wrangler job is used to process your data using the steps defined in your data flow. To learn more about all export options, see [Export \(p. 645\)](#).

### Export to Data Wrangler Job Notebook

When you export your data flow using a **Data Wrangler job**, a Jupyter Notebook is automatically created. This notebook automatically opens in your Studio instance and is configured to run a SageMaker processing job to run your Data Wrangler data flow, which is referred to as a Data Wrangler job.

1. Save your data flow. Select **File** and then select **Save Data Wrangler Flow**.
2. Choose the **Export** tab.
3. Select the last step in your data flow.  

4. Choose **Data Wrangler Job**. This opens a Jupyter Notebook.
5. Choose any **Python 3 (Data Science)** kernel for the **Kernel**.
6. When the kernel starts, run the cells in the notebook book until **Kick off SageMaker Training Job (Optional)**.

7. Optionally, you can run the cells in **Kick off SageMaker Training Job (Optional)** if you want to create a SageMaker training job to train an XGboost classifier. You can find the cost to run an SageMaker training job in [Amazon SageMaker Pricing](#).  
Alternatively, you can add the code blocks found in [Training XGBoost Classifier \(p. 587\)](#) to the notebook and run them to use the [XGBoost](#) open source library to train an XGBoost classifier.
8. Uncomment and run the cell under **Cleanup** and run it to revert the SageMaker Python SDK to its original version.

You can monitor your Data Wrangler job status in the SageMaker console in the **Processing** tab. Additionally, you can monitor your Data Wrangler job using Amazon CloudWatch. For additional information, see [Monitor Amazon SageMaker Processing Jobs with CloudWatch Logs and Metrics](#).

If you kicked off a training job, you can monitor its status using the SageMaker console under **Training jobs** in the **Training** section.

### Training XGBoost Classifier

In the same notebook that kicked off the Data Wrangler job, you can pull the data and train a XGBoost Binary Classifier using the prepared data with minimal data preparation.

1. First, upgrade necessary modules using `pip` and remove the `_SUCCESS` file (this last file is problematic when using `awswrangler`).

```
! pip install --upgrade awscli awswrangler boto sklearn
! aws s3 rm {output_path} --recursive --exclude "*" --include "*_SUCCESS"
```

2. Read the data from Amazon S3. You can use `awswrangler` to recursively read all the CSV files in the S3 prefix. The data is then split into features and labels. The label is the first column of the dataframe.

```
import awswrangler as wr

df = wr.s3.read_csv(path=output_path, dataset=True)
X, y = df.iloc[:, :-1], df.iloc[:, -1]
```

- Finally, create DMatrices (the XGBoost primitive structure for data) and do cross-validation using the XGBoost binary classification.

```
import xgboost as xgb

dmatrix = xgb.DMatrix(data=X, label=y)

params = {"objective": "binary:logistic", 'learning_rate': 0.1, 'max_depth': 5,
          'alpha': 10}

xgb.cv(
    dtrain=dmatrix,
    params=params,
    nfold=3,
    num_boost_round=50,
    early_stopping_rounds=10,
    metrics="rmse",
    as_pandas=True,
    seed=123)
```

## Shut down Data Wrangler

When you are finished using Data Wrangler, we recommend you shut down the instance it runs on to avoid incurring additional charges. To learn how to shut down the Data Wrangler app and associated instance, see [Shut Down Data Wrangler \(p. 650\)](#).

# Import

You can use Amazon SageMaker Data Wrangler to import data from the following *data sources*: Amazon Simple Storage Service (Amazon S3), Amazon Athena, Amazon Redshift, and Snowflake.

## Topics

- [Import data from Amazon S3 \(p. 588\)](#)
- [Import data from Athena \(p. 590\)](#)
- [Import data from Amazon Redshift \(p. 591\)](#)
- [Import data from Snowflake \(p. 594\)](#)
- [Imported Data Storage \(p. 617\)](#)

Some data sources allow you to add multiple *data connections*:

- You can connect to multiple Amazon Redshift clusters. Each cluster becomes a data source.
- You can query any Athena database in your account to import data from that database.

When you import a dataset from a data source, it appears in your data flow. Data Wrangler automatically infers the data type of each column in your dataset. To modify these types, select the **Data types** step and select **Edit data types**.

When you import data from Athena or Amazon Redshift, the imported data is automatically stored in the default SageMaker S3 bucket for the Amazon Region in which you are using Studio. Additionally, Athena stores data you preview in Data Wrangler in this bucket. To learn more, see [Imported Data Storage \(p. 617\)](#).

### Important

The default Amazon S3 bucket may not have the least permissive security settings like bucket policy and server-side encryption (SSE). We strongly recommend that you [Add a Bucket Policy To Restrict Access to Datasets Imported to Data Wrangler](#).

### Important

In addition, if you use the managed policy for SageMaker, we strongly recommend that you scope it down to the most restricted policy that allows you to perform your use case. For more information, see [Grant an IAM Role Permission to Use Data Wrangler](#).

## Import data from Amazon S3

Amazon Simple Storage Service (Amazon S3) can be used to store and retrieve any amount of data at any time, from anywhere on the web. You can accomplish these tasks using the Amazon Web Services Management Console, which is a simple and intuitive web interface, and the Amazon S3 API. If your dataset is stored locally, we recommend that you add it to an S3 bucket for import into Data Wrangler. To learn how, see [Uploading an object to a bucket](#) in the Amazon Simple Storage Service Developer Guide.

Data Wrangler uses [S3 Select](#) to allow you to preview your Amazon S3 files in Data Wrangler. You incur standard charges for each file preview. To learn more about pricing, see the **Requests & data retrievals** tab on [Amazon S3 pricing](#).

**Important**

If you plan to export a data flow and launch a Data Wrangler job, ingest data into a SageMaker feature store, or create a SageMaker pipeline, be aware that these integrations require Amazon S3 input data to be located in the same Amazon region.

You can browse all buckets in your Amazon account and import CSV and Parquet files using the Amazon S3 import in Data Wrangler.

When you choose a dataset for import, you can rename it, specify the file type, and identify the first row as a header.

**Important**

If you plan to import a CSV file, be aware that a single record must not exceed multiple lines, otherwise an error is thrown.

**To import a dataset into Data Wrangler from Amazon S3:**

1. If you are not currently on the **Import** tab, choose **Import**.
2. Under **Data Preparation**, choose **Amazon S3** to see the **Import S3 Data Source** view.
3. From the table of available S3 buckets, select a bucket and navigate to the dataset you want to import.
4. Select the file that you want to import. You can import CSV and Parquet files. If your dataset does not have a .csv or .parquet extension, select the data type from the **File Type** dropdown list.
5. If your CSV file has a header, select the check box next to **Add header to table**.
6. Use the **Preview** table to preview your dataset. This table shows up to 100 rows.
7. In the **Details** pane, verify or change the **Name** and **File Type** for your dataset. If you add a **Name** that contains spaces, these spaces are replaced with underscores when your dataset is imported.
8. **Enable sampling** is selected by default. If you do not uncheck this box, Data Wrangler will sample and import up to 100 MB of data. To disable sampling, uncheck this check box.
9. Choose **Import dataset**.

The screenshot shows the Data Wrangler interface with the 'Import' tab selected. On the left, a sidebar lists 'Data source / S3 Metrics / Athena test data'. Below it, a section titled 'Import a dataset from S3' provides instructions: 'Use the following table to browse S3 buckets to see import options. The following file formats are supported: CSV and Parquet.' A table lists four objects:

Object Name	Size	Last Modified
booksellers-with-categories.csv	49.94KB	2020-11-27 08:30:29+00:00
business_by_tenant_2019.csv	6.48KB	2020-11-27 08:35:19+00:00
heart_failure_clinical_records.csv	11.35KB	2020-11-27 08:33:08+00:00
titles.csv	114.98KB	2020-11-28 09:04:53+00:00

The right panel contains 'DETAILS' settings for importing 'booksellers-with-categories.csv' from 's3://sagemaker-test-data/booksellers' as a 'CSV' file. It includes checkboxes for 'Add header to rows' and 'Enable sampling'. At the bottom is a large orange 'Import CSV' button.

Below the main area, a 'Preview' section shows a sample of the imported data:

Name	Author	User Rating	Reviews	Price
10-Day Green Smoothie... A Novel	JJ Smith	4.7	17350	\$
11/22/63: A Novel	Stephen King	4.6	3652	\$2
12 Rules for Life: An Antidote to Chaos	Jordan B. Peterson	4.7	18878	\$19
1984 (Signet Classics)	George Orwell	4.7	21424	\$
5,000 Awesome Facts (L... National Geographic Kids	National Geographic Kids	4.8	7645	\$12
A Dance with Dragons (L... A Game of Thrones, #4	George R. R. Martin	4.6	13645	\$15
A Game of Thrones / A... A Song of Ice and Fire, Book 1	George R. R. Martin	4.7	19333	\$20
A Gentleman in Moscow... A Novel	Amitav Ghosh	4.7	19698	\$18
A Higher Loyalty: Truth,... A Memoir	James Comey	4.7	1983	\$
A Man Called Ove: A No... A Novel	Fredrik Backman	4.6	23846	\$
A Man Called Ove: A No... A Novel	Fredrik Backman	4.6	23846	\$

## Import data from Athena

Amazon Athena is an interactive query service that makes it easy to analyze data directly in Amazon S3 using standard SQL. With a few actions in the Amazon Web Services Management Console, you can point Athena at your data stored in Amazon S3 and begin using standard SQL to run ad-hoc queries and get results in seconds. To learn more, see [What is Amazon Athena?](#) in the Amazon Athena User Guide.

You can query Athena databases and import the results in Data Wrangler. To use this import option, you must create at least one database in Athena. To learn how, see [Getting Started](#) in the Amazon Athena User Guide.

Note the following about the Athena import option in Data Wrangler:

- Data Wrangler supports using an Athena primary workgroup. Other workgroups are not supported.
- Data Wrangler does not support federate queries.

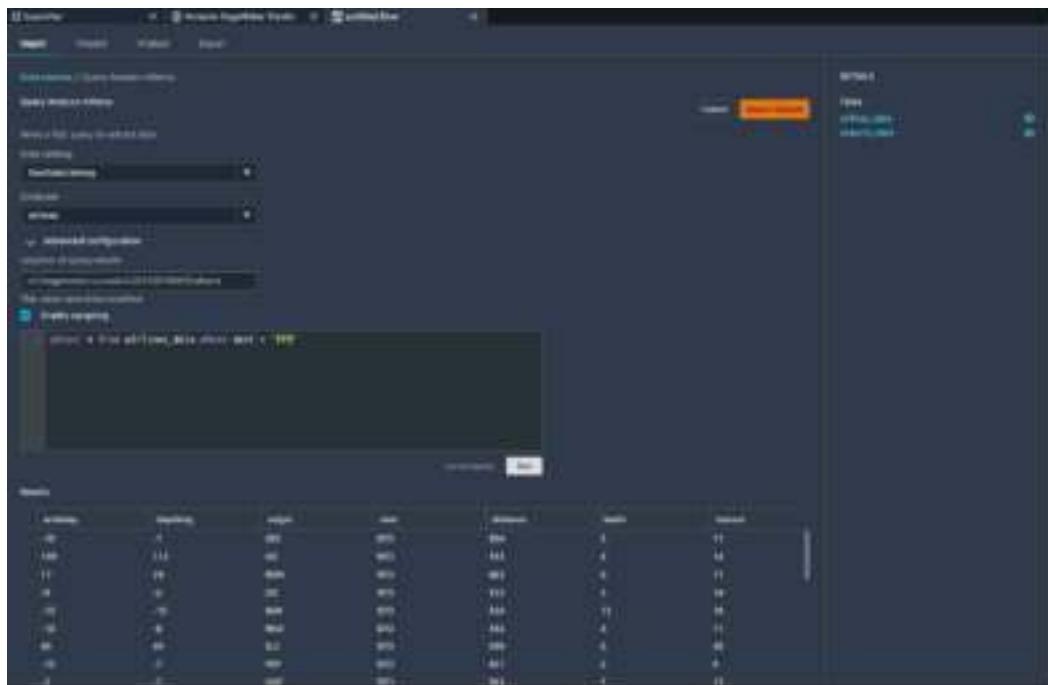
Data Wrangler uses the default S3 bucket in the same Amazon Region in which your Studio instance is located to store Athena query results. When you import from Athena, Data Wrangler creates a new database in your Athena account named `sagemaker_data_wrangler` if one does not already exist. It creates temporary tables in this database to move the query output to this S3 bucket. It deletes these

tables after data has been imported; however the database, `sagemaker_data_wrangler`, persists. To learn more, see [Imported Data Storage \(p. 617\)](#).

If you use Amazon Lake Formation with Athena, make sure your Lake Formation IAM permissions do not override IAM permissions for the database `sagemaker_data_wrangler`.

### To import a dataset into Data Wrangler from Athena:

1. On the import screen, choose **Amazon Athena**.
2. For **Catalog**, choose **AWSDataCatalog**.
3. Use the **Database** dropdown list to select the database that you want to query. When you select a database, you can preview all tables in your database using the **Tables** listed under **Details**.
4. Enter a query in the code box.
5. Under **Advanced configuration**, **Enable sampling** is selected by default. If you do not uncheck this box, Data Wrangler samples and imports approximately 50% of the queried data. Unselect this check box to disable sampling.
6. Enter your query in the query editor and use the **Run** button to run the query. After a successful query, you can preview your result under the editor.
7. To import the queried results, select **Import dataset**.
8. Enter a **Dataset name**. If you add a **Dataset name** that contains spaces, these spaces are replaced with underscores when your dataset is imported.
9. Select **Add**.



## Import data from Amazon Redshift

Amazon Redshift is a fully managed, petabyte-scale data warehouse service in the cloud. The first step to create a data warehouse is to launch a set of nodes, called an Amazon Redshift cluster. After you provision your cluster, you can upload your dataset and then perform data analysis queries.

You can connect to and query one or more Amazon Redshift clusters in Data Wrangler. To use this import option, you must create at least one cluster in Amazon Redshift. To learn how, see [Getting started with Amazon Redshift](#).

Data Wrangler uses the default S3 bucket in the same Amazon Region in which your Studio instance is located to store Amazon Redshift query results. To learn more, see [Imported Data Storage \(p. 617\)](#).

If you use the IAM managed policy, `AmazonSageMakerFullAccess`, to grant a role permission to use Data Wrangler in Studio, your **Database User** name must have the prefix `sagemaker_access`.

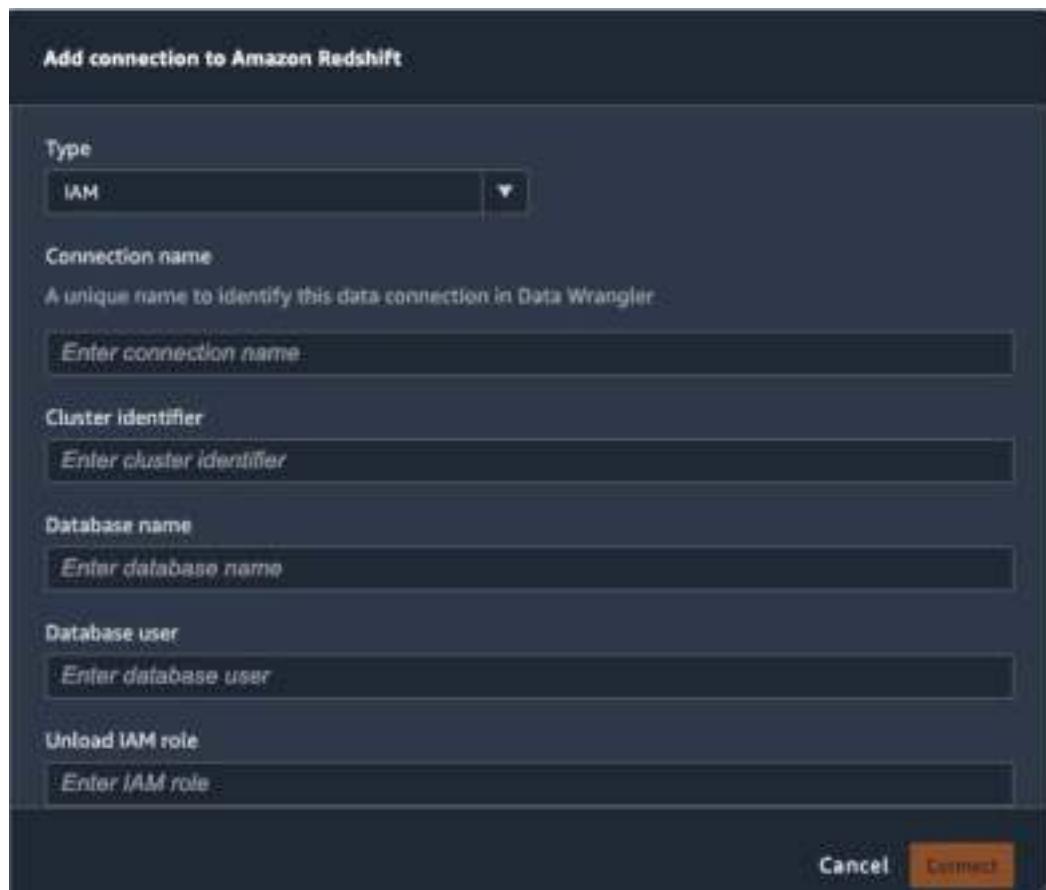
Use the following procedures to learn how to add a new cluster.

**Note**

Data Wrangler uses the Amazon Redshift Data API with temporary credentials. To learn more about this API, refer to [Using the Amazon Redshift Data API](#) in the Amazon Redshift Cluster Management Guide.

**To connect to a Amazon Redshift cluster:**

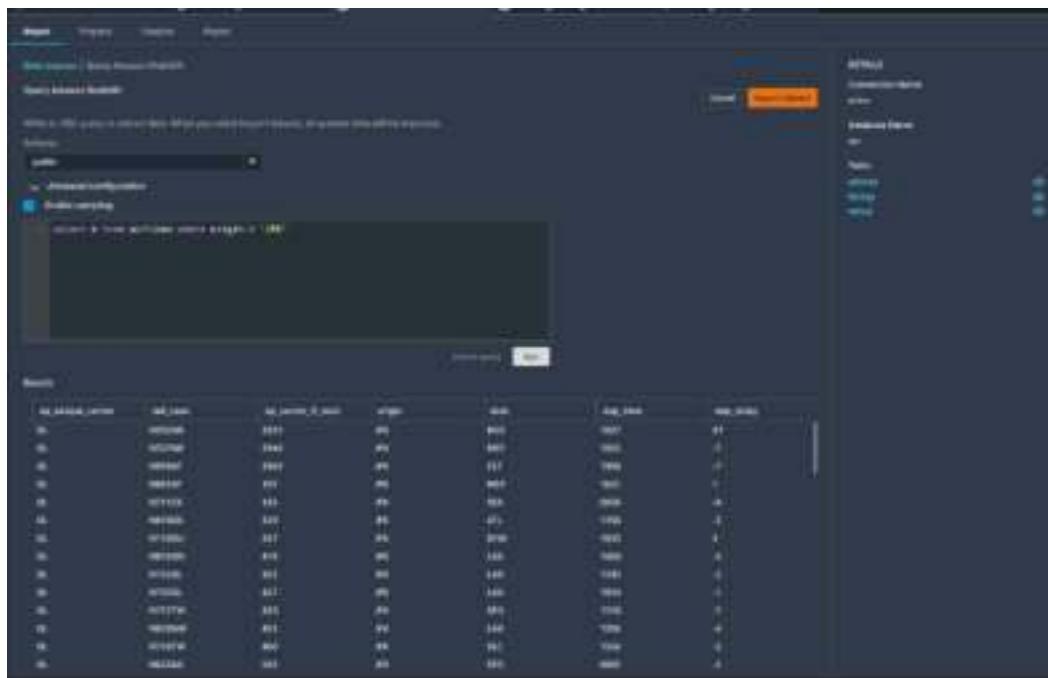
1. Choose **Import**.
2. Choose **+** under **Add data connection**.
3. Choose **Amazon Redshift**.
4. Choose **Temporary credentials (IAM)** for **Type**.
5. Enter a **Connection Name**. This is a name used by Data Wrangler to identify this connection.
6. Enter the **Cluster Identifier** to specify to which cluster you want to connect.
7. Enter the **Database Name** of the database to which you want to connect to.
8. Enter a **Database User** to identify the user you want to use to connect to the database.
9. For **UNLOAD IAM Role**, enter the IAM role ARN of the role that the Amazon Redshift cluster should assume to move and write data to Amazon S3. For more information about this role, see [Authorizing Amazon Redshift to access other Amazon services on your behalf](#) in the Amazon Redshift Cluster Management Guide.
10. Choose **Connect**.



After your connection is successfully established, it appears as a data source under **Data Import**. Select this data source to query your database and import data.

#### To query and import data from Redshift:

1. Select the connection that you want to query from **Data Sources**.
2. Select a **Schema**. To learn more about Redshift Schemas, see [Schemas](#) in the Amazon Redshift Database Developer Guide.
3. Under **Advanced configuration**, **Enable sampling** is selected by default. If you do not uncheck this box, Data Wrangler samples and imports approximately 50% of the queried data. Unselect this check box to disable sampling.
4. Enter your query in the query editor and use the **Run** button to run the query. After a successful query, you can preview your result under the editor.
5. Select **Import dataset** to import the dataset that has been queried.
6. Enter a **Dataset name**. If you add a **Dataset name** that contains spaces, these spaces are replaced with underscores when your dataset is imported.
7. Select **Add**.



## Import data from Snowflake

You can use Snowflake as a data source in SageMaker Data Wrangler to prepare data in Snowflake for machine learning.

With Snowflake as a data source in Data Wrangler, you can quickly and easily connect to Snowflake without writing a single line of code. Additionally, you can join your data in Snowflake with data stored in Amazon S3 and data queried through Amazon Athena and Amazon Redshift to prepare data for machine learning.

Once connected, you can interactively query data stored in Snowflake, easily transform data with 300+ pre-configured data transformations, understand data and identify potential errors and extreme values with a set of robust pre-configured visualization templates, and quickly identify inconsistencies in your data preparation workflow and diagnose issues before models are deployed into production. Finally, you can export your data preparation workflow to Amazon S3 for use with other SageMaker features such as SageMaker Autopilot, Amazon Feature Store and SageMaker Pipelines.

## Administrator Guide

### Important

To learn more about granular access control and best practices, see [Security Access Control](#).

This section is for Snowflake administrators who are setting up access to Snowflake from within SageMaker Data Wrangler.

### Important

Your administrator is responsible for managing and monitoring the access control within Snowflake. This includes what data a user can access, what storage integration a user can use, and what queries a user can run. Data Wrangler does not add a layer of access control with respect to Snowflake.

### Important

Note that granting monitor privileges can enable users the ability to see details within an object e.g., queries, usage within a warehouse.

## Configure Snowflake with Data Wrangler

To import data from Snowflake, Snowflake admins will need to configure access from Data Wrangler via Amazon S3.

This feature is currently not available in the opt-in regions.

To do this, follow these steps.

1. Configure access permissions for the S3 Bucket.

### Amazon Access Control Requirements

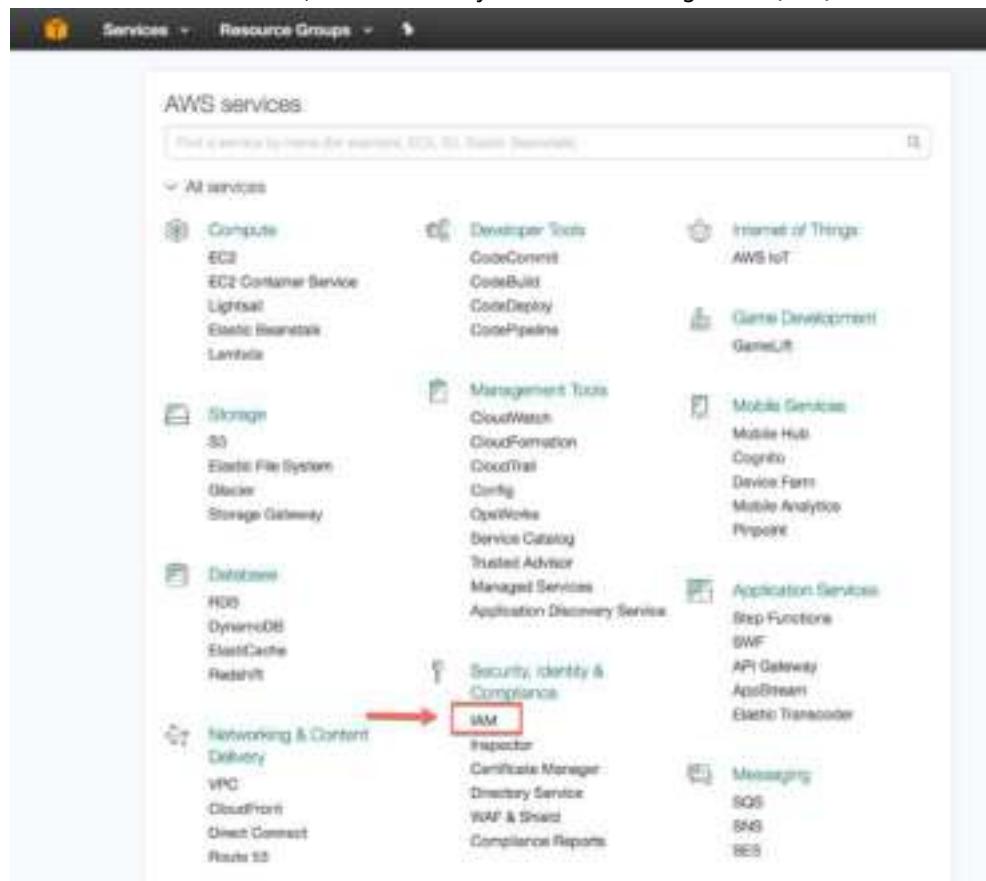
Snowflake requires the following permissions on an S3 bucket and directory to be able to access files in the directory.

- s3:GetObject
- s3:GetObjectVersion
- s3>ListBucket

### Create an IAM policy

The following describes how to configure access permissions for Snowflake in your Amazon Management Console so you can use an Amazon S3 bucket to load and unload data:

- Log in to the Amazon Management Console.
- From the home dashboard, choose Identity and Access Management (IAM):



- Choose **Policies** from the left navigation panel.
- Choose **Create Policy**:



- Select the **JSON** tab.
- Add a policy document that allows Snowflake to access the S3 bucket and directory.

The following policy (in JSON format) provides Snowflake with the required permissions to load and unload data using a single bucket and directory path. Note: Make sure to replace `bucket` and `prefix` with your actual bucket name and directory path prefix.

```
# Example policy for S3 write access
# This needs to be updated
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:GetObjectVersion",
        "s3:DeleteObject",
        "s3:DeleteObjectVersion"
      ],
      "Resource": "arn:aws:s3:::bucket/prefix/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3>ListBucket"
      ],
      "Resource": "arn:aws:s3:::bucket/",
      "Condition": {
        "StringLike": {
          "s3:prefix": ["prefix/*"]
        }
      }
    }
  ]
}
```

- Choose the button **Next: Tags**.
- Choose the button **Next: Review**.

Enter the policy name (e.g., `snowflake_access` and an optional description. Choose **Create policy**.



2. [Create the IAM Role in Amazon](#).
3. [Create a Cloud Storage Integration in Snowflake](#).
4. [Retrieve the Amazon IAM User for your Snowflake Account](#).
5. [Grant the IAM User Permissions to Access Bucket](#).
6. Grant the data scientist's Snowflake role usage permission to the storage integration.
  - In the Snowflake console, run `GRANT USAGE ON INTEGRATION integration_name TO snowflake_role;`
    - `integration_name` is the name of your storage integration.
    - `snowflake_role` is the name of the default [Snowflake role](#) given to the data scientist user.

#### What information needs to be provided to the Data Scientist

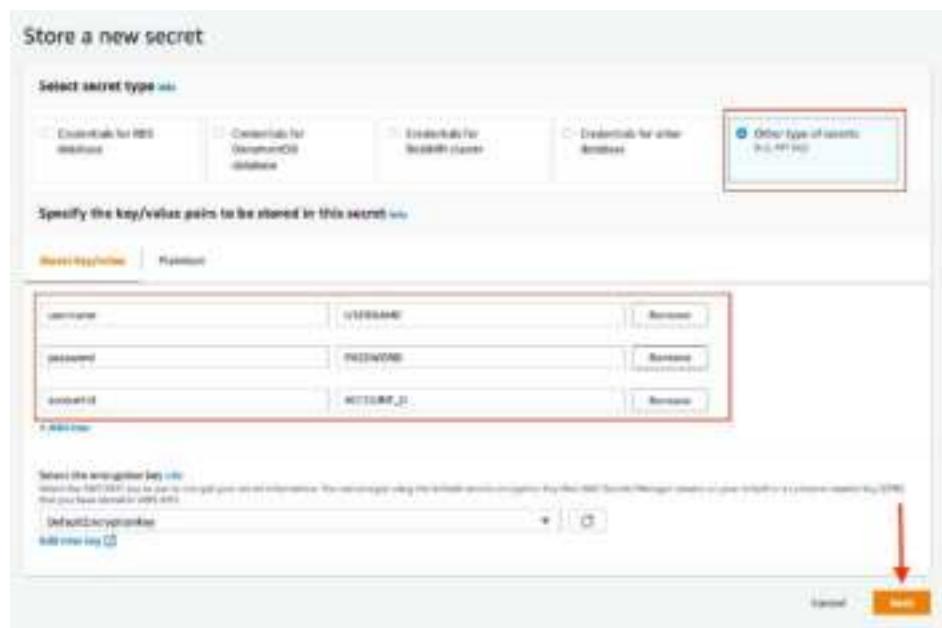
1. To allow your data scientist to access Snowflake from SageMaker Data Wrangler, provide them with one of following:
  - A Snowflake account name, user name and password.
  - Create a secret with [Amazon Secrets Manager](#) and provide the ARN of the secret. Use the following procedure below to create the secret for Snowflake if you choose this option.

#### Important

If your data scientists use the Snowflake Credentials (User name and Password) option to connect to Snowflake, note that [Secrets Manager](#) is used to store the credentials in a secret and rotates secrets as part of a best practice security plan. The secret created in Secrets Manager is only accessible with the Studio role configured when you set up Studio User profile. This will require you to add this permission, `secretsmanager:PutResourcePolicy` to the policy that is attached to your Studio role.

It is strongly recommended that you scope the role policy to use different roles for different groups of Studio users. You can add additional resource-based permissions for the Secrets Manager secrets. See [Manage Secret Policy](#) for condition keys you can use.

- [How to create an Amazon Secret for Snowflake](#).
  - Sign in to the [Amazon Secrets Manager console](#).
  - Choose **Store a new secret**.
  - In the **Select secret type** section, select **Other type of secrets**



- Specify the details of your custom secret as Key and Value pairs. The name of the keys are case sensitive and so the `username` key must be `username`, the `password` key must be `password`, and the `accountid` key must be `accountid`. If you enter any of these incorrectly, Data Wrangler will raise an error. Quotes for `username`, `password`, and `accountid` are not required if using secret key/value. Alternatively, you can select the Plaintext tab and enter the secret value in JSON as shown in the following example:

```
{
    "username": "snowflake username",
    "password": "snowflake password",
    "accountid": "snowflake accountid"
}
```

- Choose next, and on the following screen, prefix the name of your secret with `AmazonSageMaker-`. Additionally, add a tag with key `SageMaker` (without quotes) and value: `true` (without quotes). The rest of the fields are optional. You can scroll to the bottom of the page and click next. The rest of the screens are optional. Choose next until the secret has been stored.

**Secret name and description**

Secret name: **SnowflakeIntegration-DW-TEST**

Description (optional): Create for creating a new secret.

**Tags - optional**

Type: **Secret** Value: **Snowflake**

**Replicate Secret - optional**

Replicate secret to other regions

**Next Step**

- Select on the secret name and save the ARN of the secret. Next, choose the final **Store** button.
- Select the secret you just created.



- You will see your ARN on the screen. Provide the ARN to the data scientist if they are using the ARN to connect to Snowflake.



2. You will need to provide the data scientist the name of the storage integration you created in Step 3: [Create a Cloud Storage Integration in Snowflake](#). This is the name of the new integration and is called `integration_name` in the `CREATE INTEGRATION` SQL command you ran which is shown below:

```
CREATE STORAGE INTEGRATION integration_name
TYPE = EXTERNAL_STAGE
STORAGE_PROVIDER = S3
```

```
ENABLED = TRUE
STORAGE_AWS_ROLE_ARN = 'iam_role'
[ STORAGE_AWS_OBJECT_ACL = 'bucket-owner-full-control' ]
STORAGE_ALLOWED_LOCATIONS = ('s3://bucket/path/', 's3://bucket/path/')
[ STORAGE_BLOCKED_LOCATIONS = ('s3://bucket/path/', 's3://bucket/path/') ]
```

## Data Scientist Guide

This section outlines how to access your Snowflake data warehouse from within SageMaker Data Wrangler and how to use Data Wrangler features.

### Important

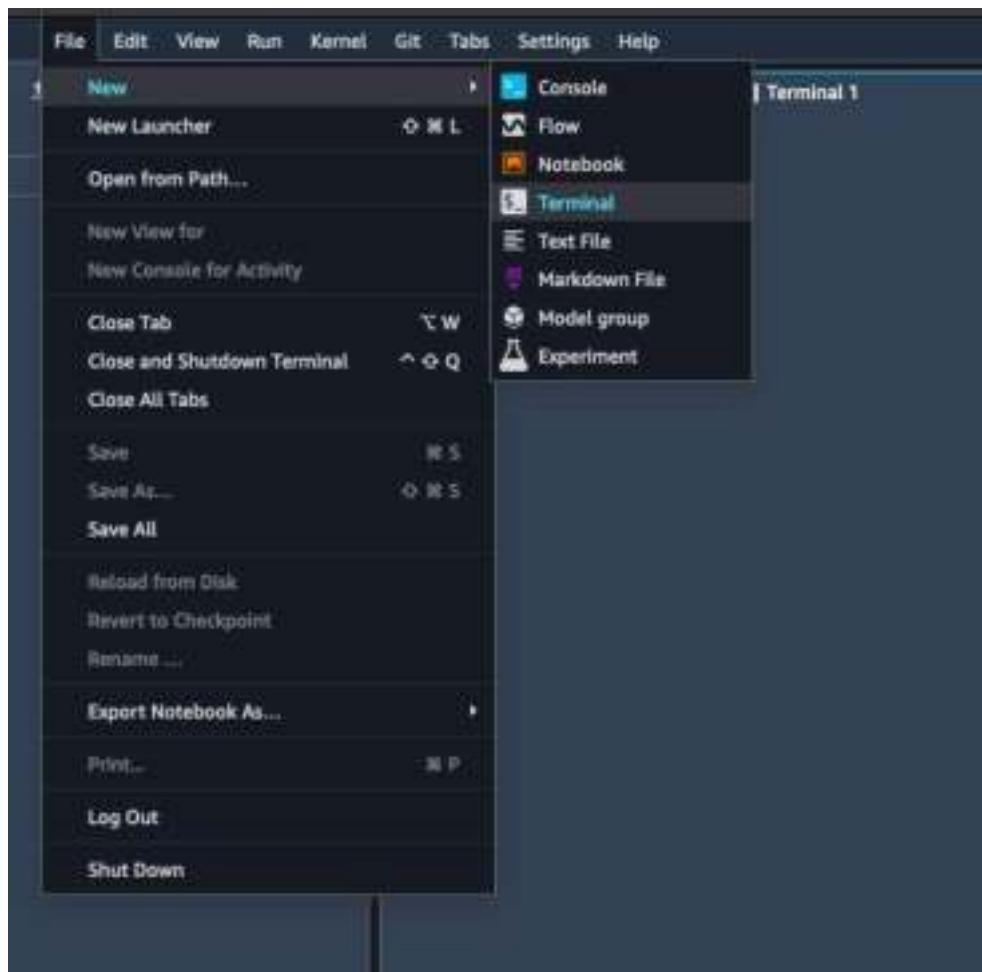
Note: Your administrator needs to follow the Administer Guide set up above before you can use Data Wrangler within Snowflake.

#### 1. Access Data Wrangler

To start, access Data Wrangler through Amazon SageMaker Studio by following the [Prerequisites](#) steps.

Once you have completed the Prerequisite steps you can now access Data Wrangler in Studio.

- Next to the user you want to use to launch Studio, select **Open Studio**.
- Once you have launched Studio, select **File** then **New** then **Terminal**.



- Enter `cat /opt/conda/share/jupyter/lab/staging/yarn.lock | grep -A 1 "@amzn/sagemaker-ui-data-prep-plugin@"` to print the version of your Studio instance. You must have Studio version 1.3.0 to enable Snowflake.

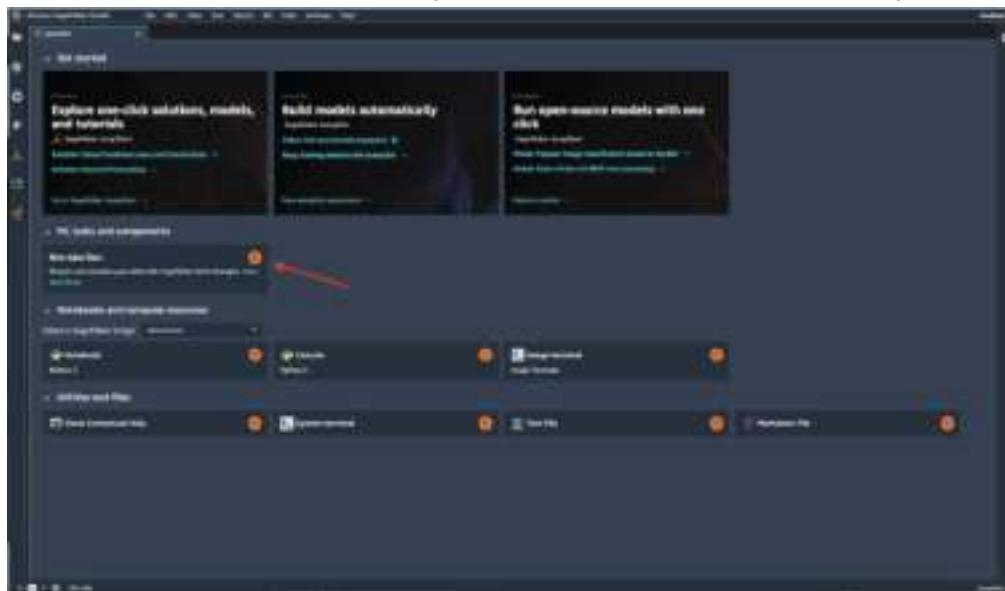
A screenshot of the "Terminal 1" window. The command `cat /opt/conda/share/jupyter/lab/staging/yarn.lock | grep -A 1 "@amzn/sagemaker-ui-data-prep-plugin@"` is run, and the output shows the package version: "version "1.3.0"" and "build "4.23"".

- If you do not have this version then update your Studio version. To do this, close your Studio window and navigate to the [SageMaker Studio Console](#).
- Next, you will select the user you are using to access Studio and then select **Delete app**. After the deletion is complete, launch Studio again by selecting **Open Studio**.

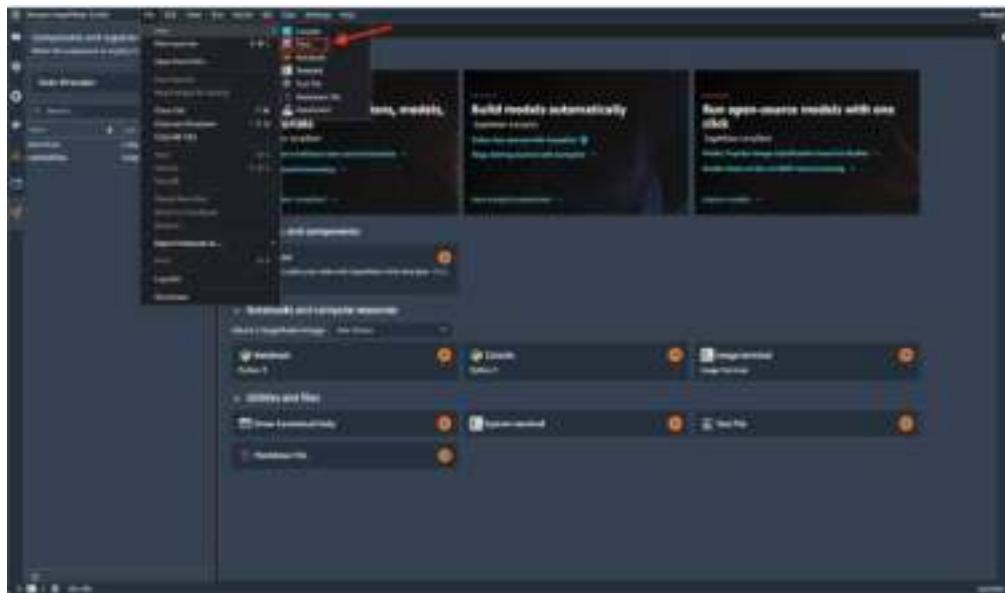


- Follow Step 3 above again to verify your Studio version is 1.3.0.
2. Create a new data flow from within Data Wrangler

Once you have accessed Data Wrangler from within Studio and have version 1.3.0, select the + sign on the **New data flow** card under **ML tasks and components**. This creates a new directory in Studio with a .flow file inside, which contains your data flow. The .flow file automatically opens in Studio.



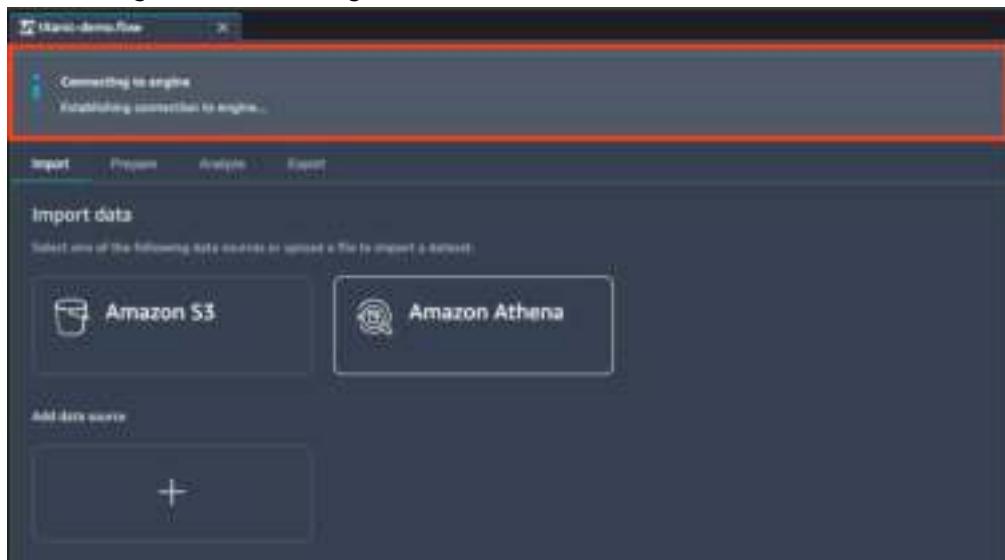
Alternatively, you can also create a new flow by selecting **File**, then **New**, and choosing **Flow** in the top navigation bar.



When you create a new .flow file in Studio, you may see a message at the top of the Data Wrangler interface that says:

**Connecting to engine**

**Establishing connection to engine...**



### 3. Connect to Snowflake

There are two ways to connect to Snowflake from within Data Wrangler. You will only need to choose one of the two ways.

1. Specify your Snowflake credentials (Account name, user name and password) in Data Wrangler.
2. Provide an Amazon Resource Name (ARN) of a secret.

**Important**

If you do not have your Snowflake credentials or ARN, reach out to your administrator. Your administrator can tell you which of the preceding two methods to use to connect to Snowflake.

Start on the **Import** data screen and first select **Add data source** from the top right corner dropdown menu and then select Snowflake. The following screenshot illustrates where to find the **Snowflake** option.



Choose an Authentication method. For this step as previously mentioned, you can use your Snowflake credentials or ARN name. One of the two will be **provided by your administrator**.

Next we explain both Authentication methods and provide screenshots for each.

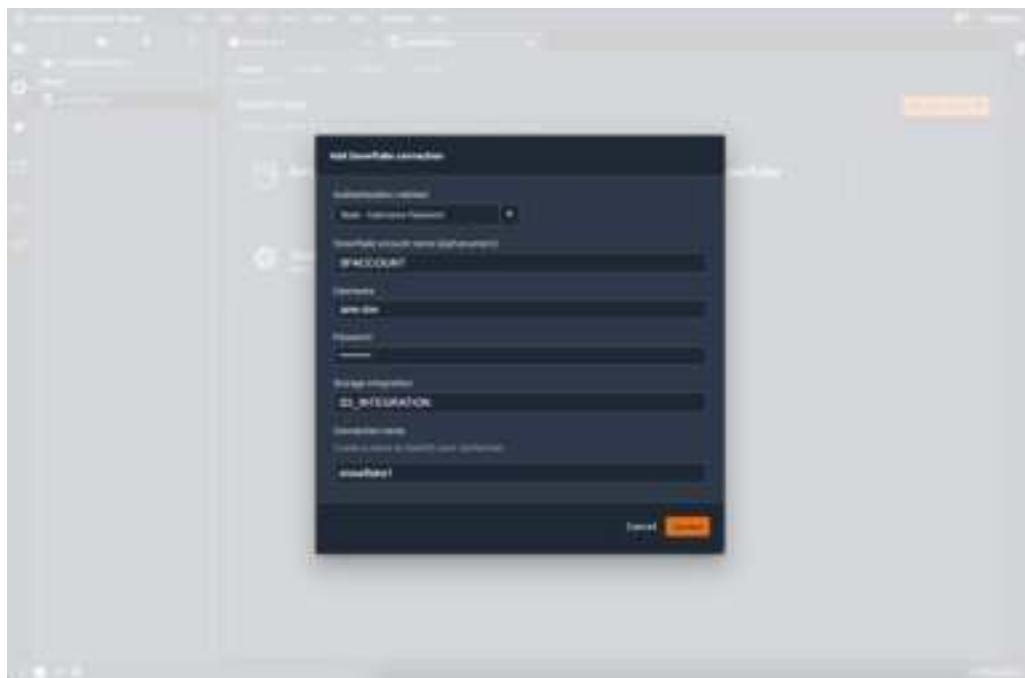
**1. Snowflake Credentials Option.**

Select the Basic (user name and password) option from the Authentication method dropdown. Then, enter your credentials to the following fields:

- **Storage integration:** Provide the name of the storage integration. This is provided from your administrator.
- **Snowflake account name:** The full name of your Snowflake account.
- **User name:** Snowflake account user name.
- **Password:** Snowflake account password.
- **Connection Name:** Choose a connection name of your choice.

Select **Connect**.

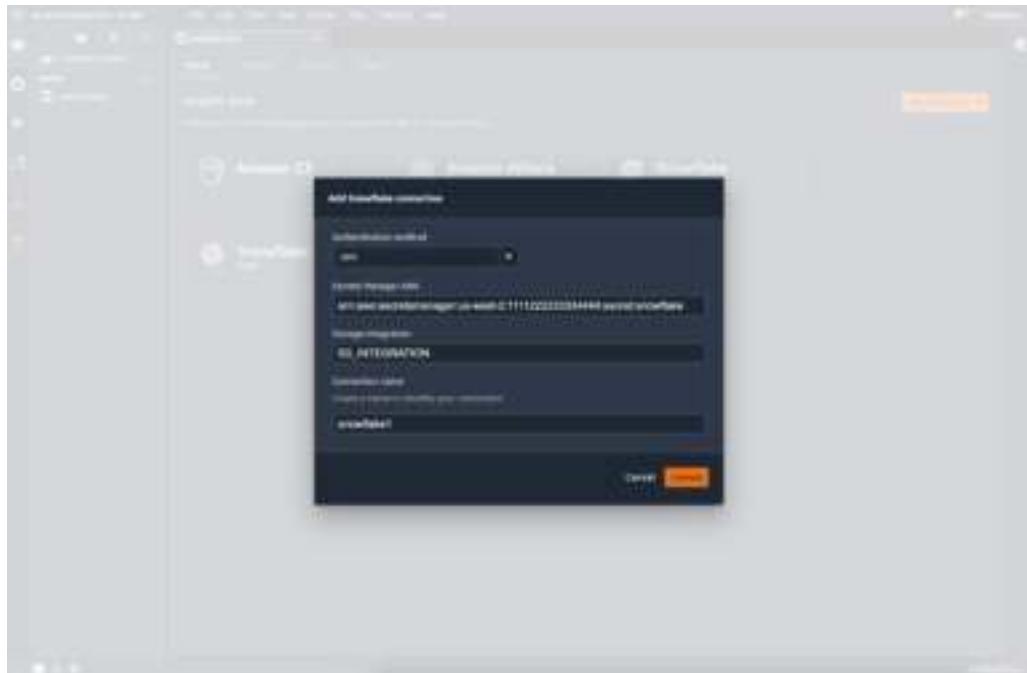
Below is a screenshot of this screen and an example on how these fields should be completed.



## 2. ARN Option

Select the ARN option under the **Authentication** method dropdown. Then, provide your ARN name under Secrets Manager ARN and your storage integration which is provided by your administrator. Lastly, create a connection name and select **Connect**.

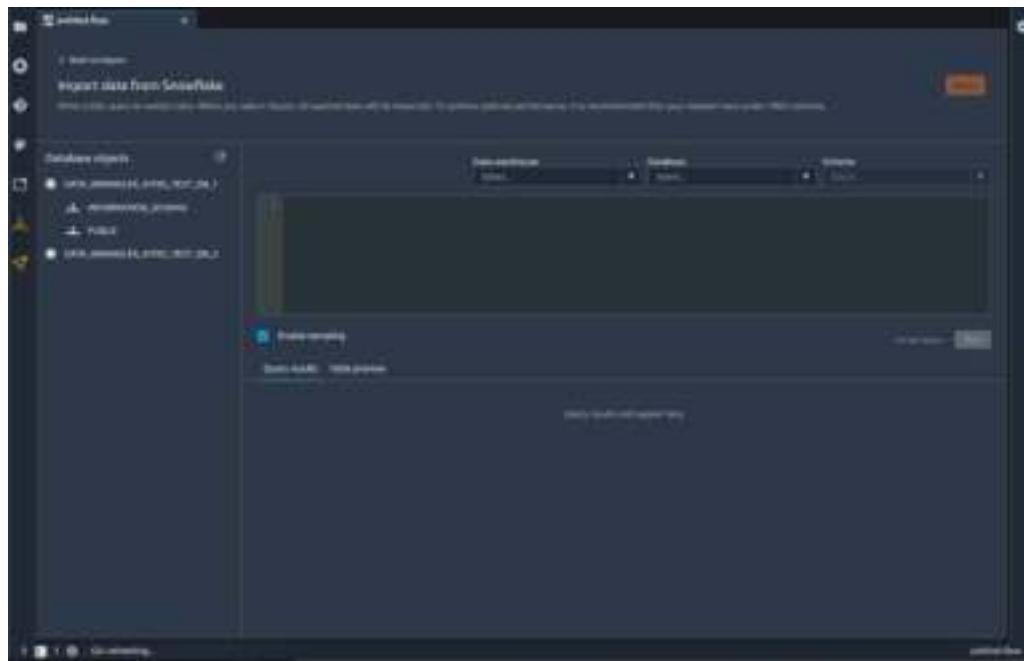
Below is a screenshot of this screen.



3. The workflow at this point is to connect your Snowflake account to Data Wrangler, then run some queries on your data and then finally use Data Wrangler for performing data transformations etc.

The next few steps explain the import and querying step from within Data Wrangler.

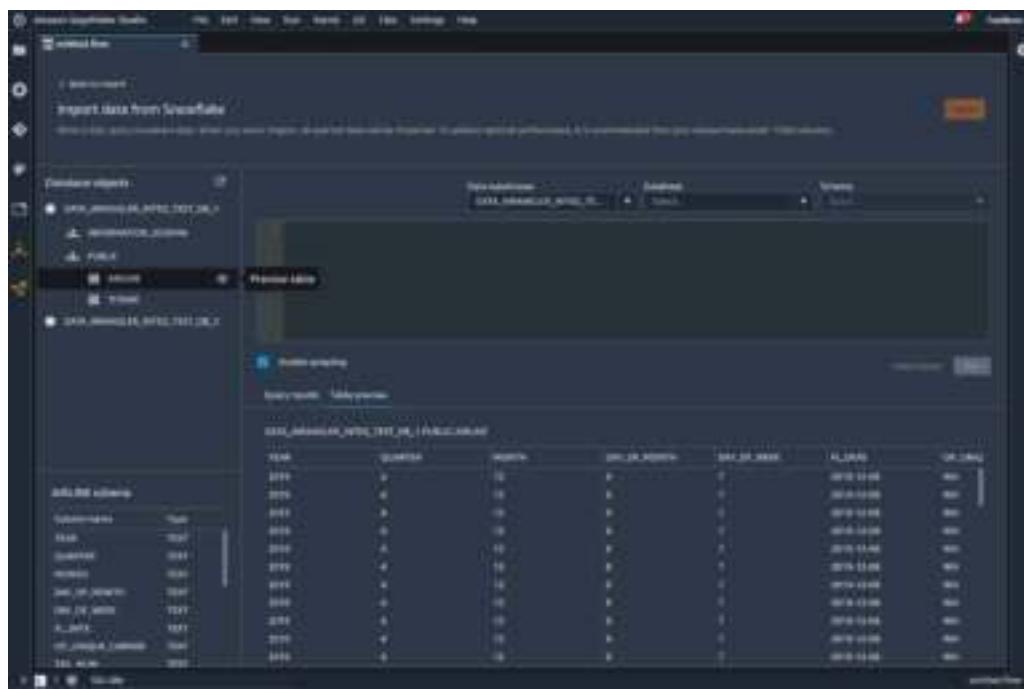
After creating your Snowflake connection you will be taken to the **Import data from Snowflake** screen. Below is a screenshot of this.



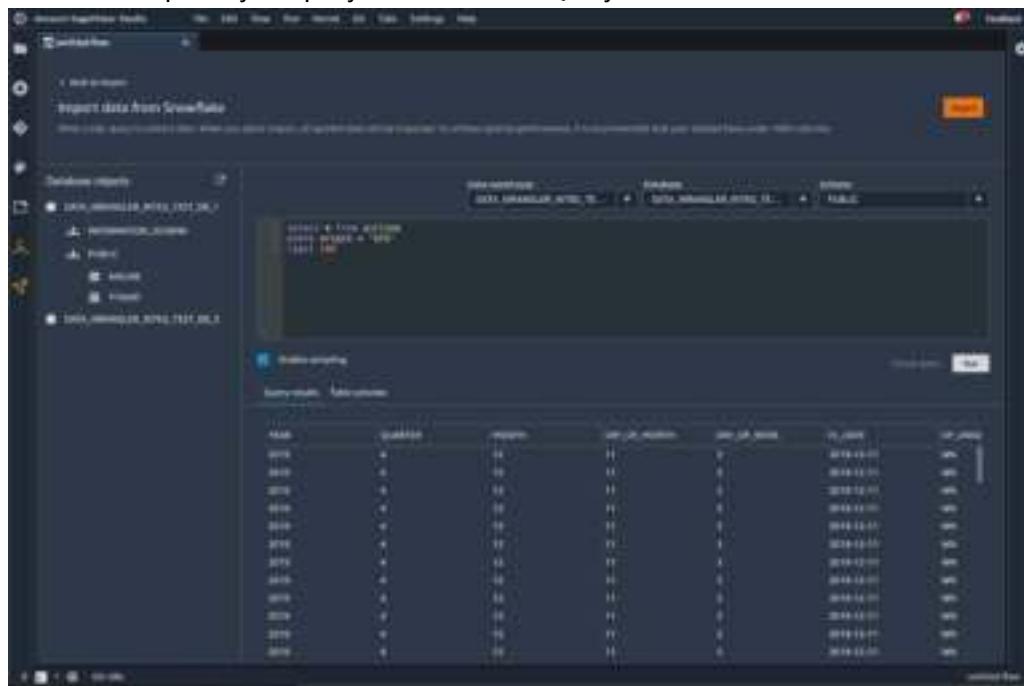
From here, select your warehouse. You can also optionally select your database and schema, in which case the written query should specify them. If **Database** and **Schema** are provided in the dropdown list, the written query does not need to specify the database and schema names.

Your schemas and tables from your Snowflake account are listed in the left panel. You can select and unravel these entities. When you select a specific table, select the eye icon on the right side of each table name to preview the table.

The following screenshot shows the panel with your data warehouses, databases, and schemas, along with the eye icon with which you can preview your table. Once you select the **Preview Table** icon, the schema preview of that table is generated. You must select a warehouse before you can preview a table.

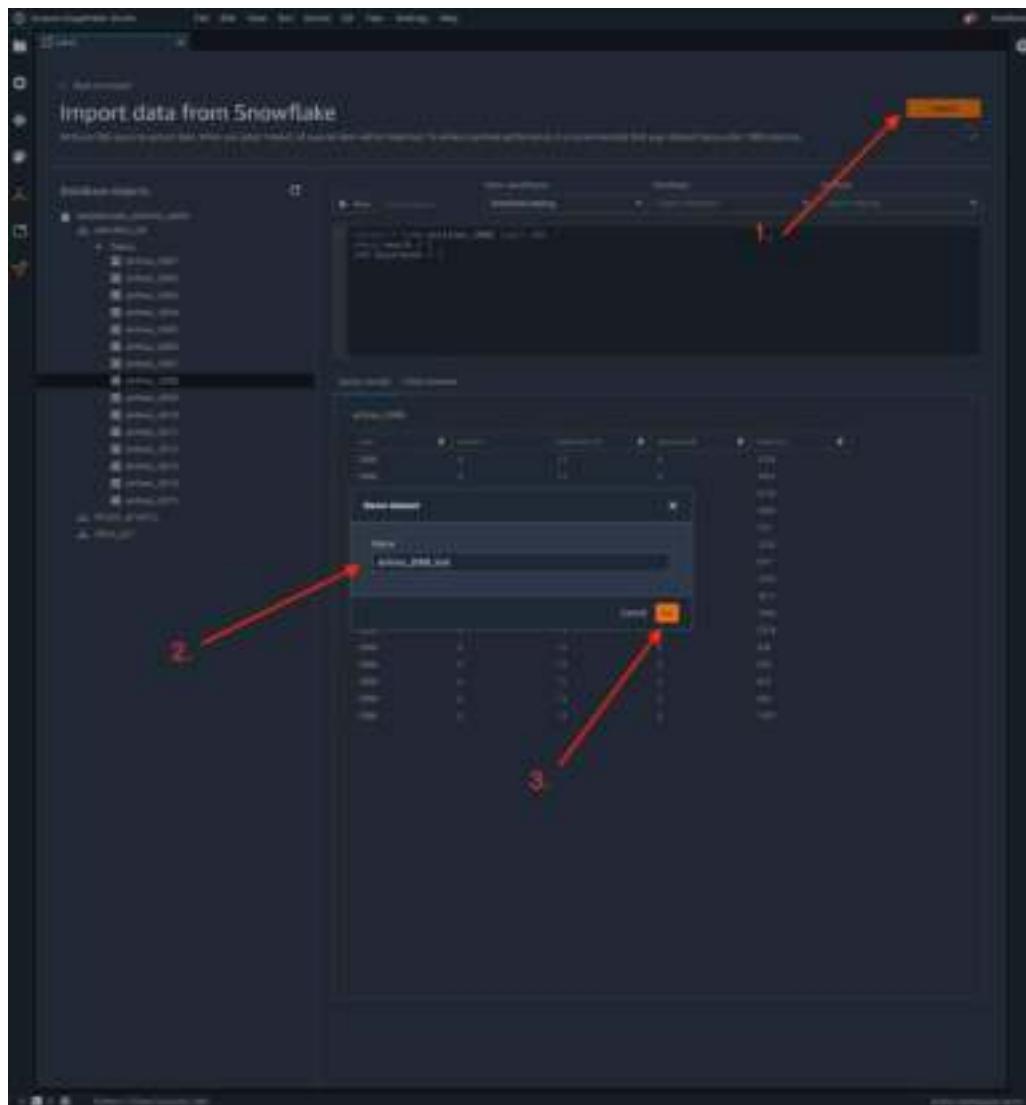


After selecting a data warehouse, database and schema, you can now write queries and select run them. The output of your query will show under Query results. Below is a screenshot of this.

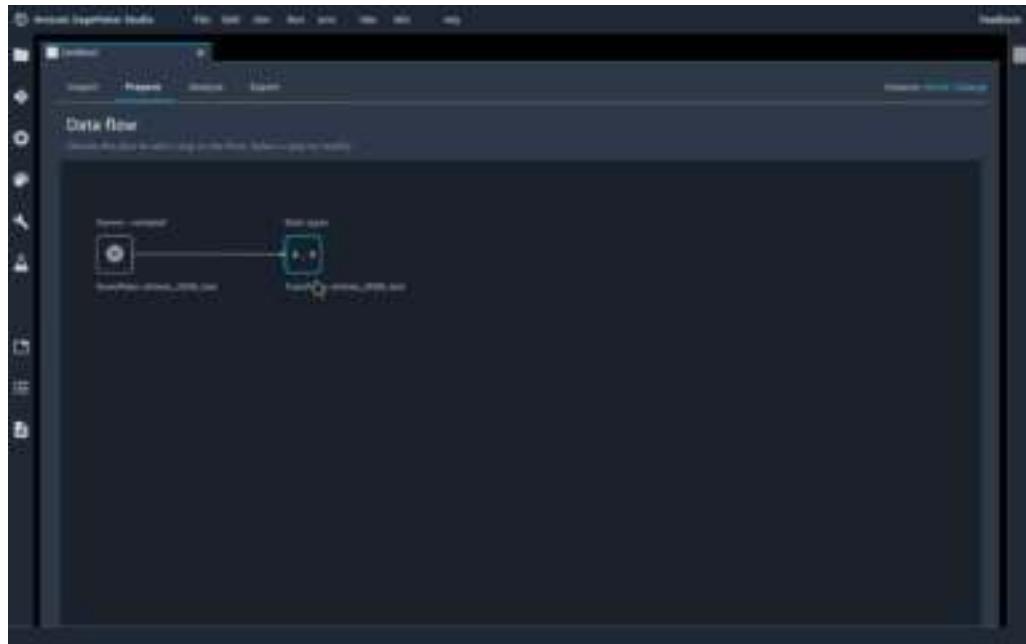


Once you have settled on the output of your query, you can then import the output of your query into a Data Wrangler flow to perform data transformations.

To do this, select **Import**, then specify a name and select **Ok**, as shown in the following screenshot.



From here, transition to the **Data flow** screen to prepare your data transformation, as shown in the following screenshot.



## Private Connectivity between Data Wrangler and Snowflake via Amazon PrivateLink

This section explains how to use Amazon PrivateLink to establish a private connection between Data Wrangler and Snowflake. The steps are explained below.

### Create a VPC

If you do not have a VPC set up, then follow the [Create a new VPC](#) instructions to create one.

Once you have a chosen VPC you would like to use for establishing a private connection, provide the following credentials to your Snowflake Administrator to enable Amazon PrivateLink:

- VPC ID.
- Amazon Account ID.
- Your corresponding account URL you use to access Snowflake.

### Important

As per Snowflake's documentation, enabling for your Snowflake account can take up to two business days.

### Set up Snowflake Amazon PrivateLink Integration

After Amazon PrivateLink is enabled, retrieve the Amazon PrivateLink configuration for your region by executing the following command in a Snowflake worksheet. Log into your Snowflake console, under worksheets enter the following: `select SYSTEM$GET_PRIVATELINK_CONFIG();`

1. Retrieve the values for the following: `privatelink-account-name`, `privatelink_ocsp-url`, `privatelink-account-url`, and `privatelink_ocsp-url` from the resulting JSON object. Examples of each value are below. Store these values for later use.

```
privatelink-account-name: xxxxxxxx.region.privatelink
```

```
privatelink-vpce-id: com.amazonaws.vpce.region.vpce-svc-xxxxxxxxxxxxxxxxxx
privatelink-account-url: xxxxxxxx.region.privatelink.snowflakecomputing.com
privatelink_ocsp-url: ocsp.xxxxxxx.region.privatelink.snowflakecomputing.com
```

2. Switch to your Amazon Console and navigate to the VPC menu.
  3. From the left side panel, click the link for "Endpoints" to navigate to VPC Endpoints set up.
- Once there, click the **Create Endpoint** button in the top left.
4. Select the radio button for "Find service by name".

A VPC endpoint enables you to privately connect your VPC to another service. There are three types of VPC endpoints – interface endpoints, gateway endpoint, and private endpoints. Interface endpoints and Gateway Load Balancer endpoints are powered by AWS PrivateLink, and use an elastic network interface (ENI) as an entry point for traffic destined to the service. Interface endpoints are typically addressed using the public or private DNS name associated with the service, while gateway endpoints and Gateway Load Balancer endpoints can be targeted by a route in your route table for traffic destined for the service.

Service category:  AWS services    Find service by name    Your AWS Marketplace services

Service Name: Enter private service name and verify:

**Verify**

5. In the Service Name field, paste in the value for **privatelink-vpce-id** that you retrieved in the step above and select the **Verify** button.

If successful a green alert with "Service name found" will appear on your screen and the VPC and Subnet options will automatically expand. Below is a screenshot showing this. Note: Depending on your targeted region, your resulting screen may show another Amazon region name.

A VPC endpoint enables you to privately connect your VPC to another service. There are three types of VPC endpoints – interface endpoints, gateway Load Balancer endpoints, and private endpoints. Interface endpoints and Gateway Load Balancer endpoints are powered by AWS PrivateLink, and use an elastic network interface (ENI) as an entry point for traffic destined to the service. Interface endpoints are typically addressed using the public or private DNS name associated with the service, while gateway endpoints and Gateway Load Balancer endpoints can be targeted by a route in your route table for traffic destined for the service.

Service category:  AWS services    Find service by name    Your AWS Marketplace services

Service Name: Enter private service name and verify:

**Service name found.**

**Verify**

VPC:

Subnet:

Availability Zone	Subnet ID
<input checked="" type="checkbox"/> us-west-2a (subnet-0123)	subnet-01234567
<input checked="" type="checkbox"/> us-west-2b (subnet-0234)	subnet-02345678
<input checked="" type="checkbox"/> us-west-2c (subnet-0345)	subnet-03456789

6. Select the same VPC ID that you sent to Snowflake from the VPC dropdown menu.
  7. If you have not yet created a subnet, then following the next set of instructions on how to do this.
  8. Select the Subnets from the VPC dropdown menu. Then select Create subnet and follow the prompts to create a subset in your VPC. Ensure you select the VPC ID you sent Snowflake.
  9. Scroll down to the Security Group Configuration, select Create New Security Group. This will open the default Security Group screen in a new tab. In this new tab, select the Create Security Group button in the top right corner.
10. Provide a new security group a name (e.g., datawrangler-doc-snowflake-privatelink-connection) and a description. Be sure to select the VPC ID you have used in previous steps.

11. You now will need to add two rules to allow traffic from within your VPC to this VPC endpoint.

    Navigate to your VPC under Your VPCs in a separate tab, and retrieve your CIDR block for your VPC. Then select the Add Rule button in the Inbound Rules section in Select HTTPS for the type, leave the Source as Custom in the form, paste in the value retrieved from the above describe-vpcs call (e.g., 10.0.0.0/16).

12. Select the Create Security Group button in the bottom right. Retrieve the Security Group ID from the newly created security group (e.g., sg-xxxxxxxxxxxxxxxxxxxx).

13. In the VPC Endpoint configuration screen, remove the default security group. Paste in the security group ID in the search field and select the check box.



14. Scroll down to the bottom of the page and select the Create Endpoint button.

15. On success you will be navigated to page that has a link to your VPC Endpoint configuration, specified by the VPC ID. Select the link to view the configuration in full.



Retrieve the topmost record in the DNS names list. This can be differentiated from other DNS names because it will only include the region name (e.g., us-west-2), and no availability zone letter notation (e.g., us-west-2a). Store this information for later use.

## Configure DNS for Snowflake Endpoints in your VPC

This section explains how to configure DNS for Snowflake Endpoints in your VPC. This will allow your VPC to resolve requests to the Snowflake Amazon PrivateLink endpoint.

1. Navigate to the [Route 53 menu](#) within your Amazon console.
2. From the left hand menu (this might need to be expanded) select the Hosted Zones option.
3. Select the Create Hosted Zone button in the top right.
  - In the domain name form field, reference the value that was stored for `privatelink-account-url` in the steps above. In this field, your Snowflake account ID will be removed from the DNS name and will only use the value starting with the region identifier. A Resource Record Set will also be created later for the subdomain. E.g., `region.privatelink.snowflakecomputing.com`.
  - Select the radio button for **Private Hosted Zone** in the Type Section. Your region code may not be us-west-2. Reference the DNS name returned to you by Snowflake.

### Create hosted zone Info

**Hosted zone configuration**

A hosted zone is a container that holds information about how you want to route traffic for a domain, such as example.com, and its subdomains.

**Domain name Info**  
 This is the name of the domain that you want to route traffic to.  
  
 Valid characters: a-z, 0-9, ., #, \$, %, &, !, (, ), \*, +, =, ;, :, /, @, [ , ], \, ^, ~  
**Description - optional Info**  
 This value lets you distinguish hosted zones that have the same name.  
  
 The description can have up to 256 characters. 67/256  
**Type Info**  
 The type indicates whether you want to route traffic on the internet or in an Amazon VPC.  
 **Public hosted zone**  
 A public hosted zone determines how traffic is routed on the internet.  
 **Private hosted zone**  
 A private hosted zone determines how traffic is routed within an Amazon VPC.

- In the VPCs to associate with the hosted zone section, select the region in which your VPC is located and the VPC ID used in previous steps.

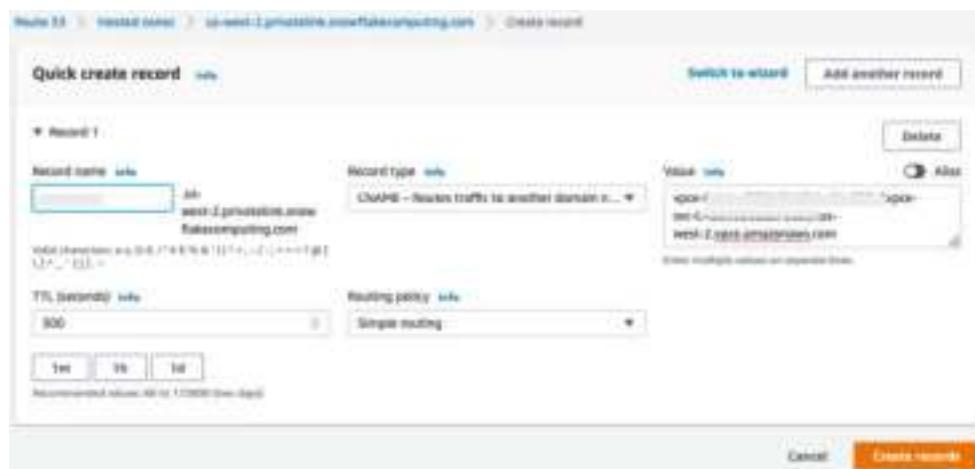
### VPCs to associate with the hosted zone Info

To use this hosted zone to resolve DNS queries for one or more VPCs, choose the VPCs. To associate a VPC with a hosted zone when the VPC was created using a different AWS account, you must use a programmatic method, such as the AWS CLI.

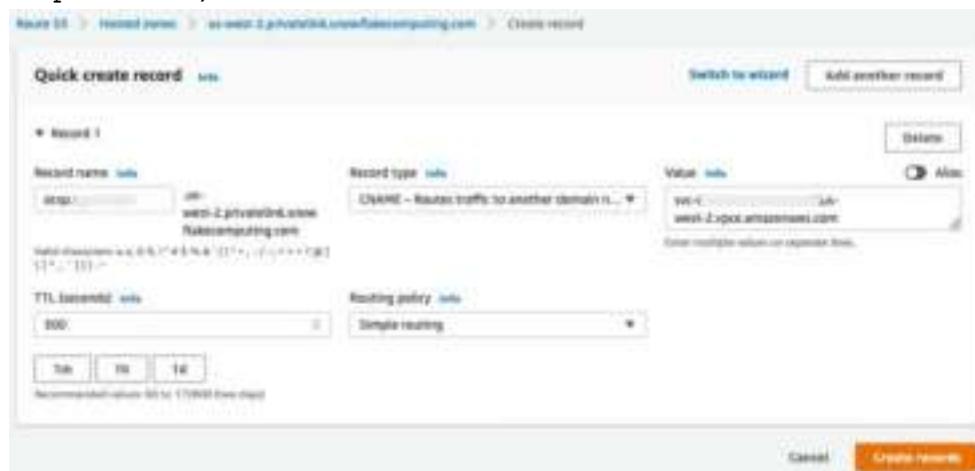
(1) For each VPC that you associate with a private hosted zone, you must set the Amazon VPC settings `enableDnsHostnames` and `enableDnsSupport` to true.

<b>Region <small>Info</small></b>	<b>VPC ID <small>Info</small></b>
US West (Oregon) (us-west-2)	<input type="text" value="Q_vpc-"/> <input type="button" value="Remove VPC"/>
<input type="button" value="Add VPC"/>	

- Select the Created hosted zone button in the bottom right
- Next we will create two records, one for `privatelink-account-url` and one for `privatelink_ocsp-url`.
    - Within the Hosted Zone menu, select the Create Record Set button.
    - For the record name, enter your Snowflake Account ID only (the first 8 characters in `privatelink-account-url`).
    - For record type, select CNAME.
    - For value, enter the DNS name for the regional VPC Endpoint you retrieved in the last step of the Set up the Snowflake Amazon PrivateLink Integration section.



- d. Select the Create Record button.
- e. Repeat the above steps for the OCSP record we notated as `privatelink-ocsp-url` earlier, starting with "ocsp" through the 8 character Snowflake ID for the record name (e.g., `ocsp .xxxxxxxxx`).



## Configure Route 53 Resolver Inbound Endpoint for your VPC

This section explains how to configure Route 53 resolvers inbound endpoints for your VPC.

1. Navigate to the [Route 53 menu](#) within your Amazon console.
  - In the left hand panel in the security section, select the Security Groups option.
2. Select the Create Security Group button in the top right corner.
  - Provide your security group a name (e.g.,: `datawranger-doc-route53-resolver-sg`) and description.
  - Select the VPC ID used in previous steps.
  - Create rules that allow for DNS over UDP and TCP from within the VPC CIDR block .



- Select the **Create Security Group** button. Note the Security Group ID because it will now add a rule to allow traffic to the VPC Endpoint Security Group.
3. Navigate to the [Route 53 menu](#) within your Amazon console.
    - In the left hand panel in the resolver section, select the Inbound Endpoint option.
  4. Select the **Create Inbound Endpoint** button.
    - Provide an endpoint a name.
    - In the VPC in the Region dropdown, select the VPC ID you have used in all previous steps.
    - In the Security Group dropdown, select the security group ID from Step 2 in this section.

The screenshot shows the 'General settings for inbound endpoint' configuration page. It includes fields for 'Endpoint name' (set to 'moonmaxw-resolver'), 'VPC in the Region' (set to 'us-west-2 (Oregon)'), and 'Security group for this endpoint' (set to 'moonmaxw-r53-resolver-qsg-1').

**General settings for inbound endpoint**

**Endpoint name**  
A friendly name lets you easily find your endpoint on the dashboard.  
moonmaxw-resolver

The endpoint name can have up to 64 characters. Valid characters: a-z, A-Z, 0-9, space, \_, (underscore), and - (hyphen).

**VPC in the Region:** us-west-2 (Oregon) [Info](#)  
All inbound DNS queries will flow through this VPC on the way to Resolver. You can't change this value after you create an endpoint.

vpc-1 (moonmaxw-vpc)

**Security group for this endpoint**  
A security group controls access to this VPC. The security group that you choose must include one or more inbound rules. You can't change this value after you create an endpoint.

moonmaxw-r53-resolver-qsg-1

- In the IP Addresses Section, select two Availability Zones, subnets and leave the radio selector for **Use an IP address that is selected automatically** selected.

**▼ IP address #1**

[Remove IP address](#)

**Availability Zone info**  
The Availability Zone that you choose for inbound DNS queries must be configured with a subnet.

us-west-2a

**Subnet info**  
The subnet that you choose must have an available IP address. Only IPv4 addresses are supported.

subnet- (10.0.1.0 - us-west-2a) [10.0.1.0 ...]

**IP address info**  
For inbound DNS queries, you can either let the service choose an IP address for you from the available IP addresses in the subnet, or you can specify the IP address yourself.

Use an IP address that is selected automatically  
 Use an IP address that you specify

**▼ IP address #2**

[Remove IP address](#)

**Availability Zone info**  
The Availability Zone that you choose for inbound DNS queries must be configured with a subnet.

us-west-2c

**Subnet info**  
The subnet that you choose must have an available IP address. Only IPv4 addresses are supported.

subnet- (10.0.3.0 - us-west-2c) [10.0.3.0 ...]

**IP address info**  
For inbound DNS queries, you can either let the service choose an IP address for you from the available IP addresses in the subnet, or you can specify the IP address yourself.

Use an IP address that is selected automatically  
 Use an IP address that you specify

[Add another IP address](#)

- Select the Submit button.
- Select the Inbound Endpoint after it was created.
  - Once the Inbound Endpoint is created, note the two IP addresses for the resolvers.



The screenshot shows a table of VPC endpoint configurations. The columns are labeled: IP address, IP address ID, Name, Status, and Availability Zone. There are two entries:

IP address	IP address ID	Name	Status	Availability Zone
10.0.1.0	ip-10-0-1-0	resolver-1	Active	us-west-2
10.0.1.0	ip-10-0-1-0	resolver-2	Active	us-west-2

## SageMaker VPC Endpoints

This section explains how to create VPC endpoints for the following: **SageMaker Studio**, **SageMaker Notebook**, **SageMaker API**, **SageMaker Runtime**, **SageMaker FeatureStore Runtime**.

### Create a security group that will be applied to all endpoints.

1. Navigate to the [EC2 menu](#) in the Amazon Console.
2. From the left hand panel, select the Security Groups option, in the Network & Security section.
3. Select the Create Security Group button in the upper right hand corner.

4. Provide your security group a name and description (e.g., `datawrangler-doc-sagemaker-vpc-  
sg`). **Note that a rule will be added to allow traffic over HTTPS from SageMaker to this group later.**

### Creating the endpoints.

1. Navigate to the **VPC menu** in the Amazon console.
2. Select the **Endpoints** option from the left hand panel.
3. Select the **Create Endpoint** button.
4. Search for the service in the search bar (e.g., enter “sagemaker” in the search bar).
5. From the **VPC** dropdown, select the VPC in which your Snowflake Amazon PrivateLink connection exists.
6. In the **Subnets** section, select the Subnets which have access to the Snowflake PrivateLink connection.
7. Leave the **Enable DNS Name** checkbox ticked.
8. In the **Security Groups** section, select the security group you created in the section above.
9. Select the **Create Endpoint** button.

### Configure SageMaker Studio and SageMaker Data Wrangler

This section explains how to configure SageMaker Studio and SageMaker Data Wrangler.

1. Configure Security Group
  - Navigate to the EC2 menu in the Amazon Console.
  - From the left hand panel, select the Security Groups option, in the Network & Security section.
  - Select the **Create Security Group** button in the upper right hand corner.
  - Provide your security group a name and description (e.g., `datawrangler-doc-sagemaker-  
studio`).
  - Create the following inbound rules.
    - HTTPS to the Security Group you provisioned for the Snowflake PrivateLink connection.
      - As created in Set up the Snowflake PrivateLink Integration step.
    - HTTP to the Security Group you provisioned for the Snowflake PrivateLink connection.
      - As created in Set up the Snowflake PrivateLink Integration step.
    - UDP and TCP for DNS (port 53) to Route 53 Resolver Inbound Endpoint security group.
      - As created in Step 2 of **Configure Route 53 Resolver Inbound Endpoint for your VPC..**
  - Select the **Create Security Group** button in the lower right hand corner.
2. Configure SageMaker Studio
  - Navigate to the SageMaker menu in the Amazon console.
  - From the left hand console, Select the **SageMaker Studio** option.
  - If you do not have any domains configured, the Get Started menu is present.
  - Select the **Standard Setup** option from the Get Started menu.
  - For **Authentication method**, select Amazon Identity and Access Management (IAM).
  - Under the **Permissions** menu you can create a new role or use a pre-existing role, depending on your use case.
    - If you chose **Create a new role** you are presented the option to provide an S3 bucket name, and a policy is generated for you.
    - If you already have a role created with permissions to the S3 buckets you require access to, select the role from the dropdown. This role should have the **AmazonSageMakerFullAccess** policy attached to it.

- Select the **Network and Storage** dropdown to configure the VPC, Security, and Subnets SageMaker will use.
    - For **VPC** select the VPC in which your Snowflake PrivateLink connection exists.
    - For **Subnet(s)** select the Subnets which have access to the Snowflake PrivateLink connection.
    - For **Network Access for Studio** select VPC Only.
    - For **Security Group(s)** select the security group you created earlier in Step 1.
  - Select the **Submit** button in the bottom right hand corner.
3. Edit the SageMaker Security Group.
- Create the following inbound rules:
    - Port 2049 to the inbound and outbound NFS Security Groups created automatically by SageMaker in Step 2 (the security group names will contain the Studio domain ID).
    - Access to all TCP ports to itself (required for SageMaker for VPC Only).
4. Edit the VPC Endpoint Security Groups:
- Navigate to the EC2 menu in the Amazon console.
  - Locate the Security Group you created earlier.
  - Add an inbound rule allowing for HTTPS traffic from the security group created in Step 1.
5. Create a user profile.
- From the **SageMaker Studio Control Panel** select the **Add User** button in the top right.
  - Provide a user name.
  - For the **Execution Role**, choose to create a new role or to use a pre-existing role.
    - If you chose **Create a new role** you are presented the option to provide an Amazon S3 bucket name, and a policy is generated for you.
    - If you already have a role created with permissions to the Amazon S3 buckets you require access to, select the role from the dropdown. This role should have the **AmazonSageMakerFullAccess** policy attached to it.
  - Select the **Submit** button.
6. Create a Data Flow (follow the Data Scientist guide outlined above).
- When adding a Snowflake connection enter the value of **privatelink-account-name** (from Step **Set up Snowflake PrivateLink Integration**) into the Snowflake account name (alphanumeric) field, instead of the plain Snowflake account name. Everything else is left unchanged.

## Imported Data Storage

### Important

We strongly recommend that you follow the best practices around protecting your Amazon S3 bucket by following [Security best practices](#).

When you query data from Amazon Athena or Amazon Redshift, the queried dataset is automatically stored in Amazon S3. Data is stored in the default SageMaker S3 bucket for the Amazon Region in which you are using Studio.

Default S3 buckets have the following naming convention: `sagemaker-region-account number`. For example, if your account number is 111122223333 and you are using Studio in us-east-1, your imported datasets are stored in `sagemaker-us-east-1-111122223333`.

Data Wrangler flows depend on this Amazon S3 dataset location, so you should not modify this dataset in Amazon S3 while you are using a dependent flow. If you do modify this S3 location, and you want to continue using your data flow, you must remove all objects in `trained_parameters` in your `.flow` file. To do this, download the `.flow` file from Studio and for each instance of `trained_parameters`, delete all entries. When you are done, `trained_parameters` should be an empty JSON object:

```
"trained_parameters": {}
```

When you export and use your data flow to process your data, the .flow file you export refers to this dataset in Amazon S3. Use the following sections to learn more.

## Amazon Redshift Import Storage

Data Wrangler stores the datasets that result from your query in a Parquet file in your default SageMaker S3 bucket.

This file is stored under the following prefix (directory): redshift/*uuid*/data/, where *uuid* is a unique identifier that gets created for each query.

For example, if your default bucket is sagemaker-us-east-1-111122223333, a single dataset queried from Amazon Redshift is located in s3://sagemaker-us-east-1-111122223333/redshift/*uuid*/data/.

## Amazon Athena Import Storage

When you query an Athena database and import a dataset, Data Wrangler stores the dataset, as well as a subset of that dataset, or *preview files*, in Amazon S3.

The dataset you import by selecting **Import dataset** is stored in Parquet format in Amazon S3.

Preview files are written in CSV format when you select **Run** on the Athena import screen, and contain up to 100 rows from your queried dataset.

The dataset you query is located under the prefix (directory): athena/*uuid*/data/, where *uuid* is a unique identifier that gets created for each query.

For example, if your default bucket is sagemaker-us-east-1-111122223333, a single dataset queried from Athena is located in s3://sagemaker-us-east-1-111122223333/redshift/*uuid*/data/*example\_dataset.parquet*.

The subset of the dataset that is stored to preview dataframes in Data Wrangler is stored under the prefix: athena/.

## Create and Use a Data Wrangler Flow

Use an Amazon SageMaker Data Wrangler flow, or a *data flow*, to create and modify a data preparation pipeline. The data flow connects the datasets, transformations, and analyses, or *steps*, you create and can be used to define your pipeline.

## Instances

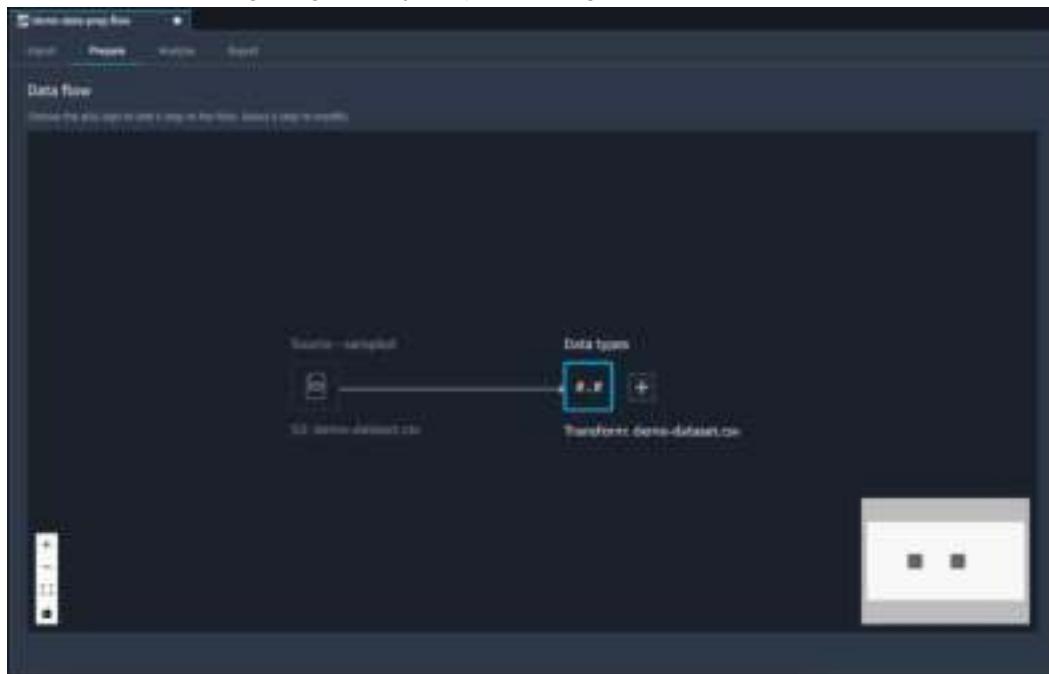
When creating a Data Wrangler flow you can select an ml.m5 instance from the following table:

Standard Instances	vCPU	Memory
ml.m5.4xlarge	16	64 GiB
ml.m5.12xlarge	48	192 GiB
ml.m5.24xlarge	96	384 GiB

For more information about the cost per hour for using the available instance types, see [SageMaker Pricing](#).

## The Data Flow UI

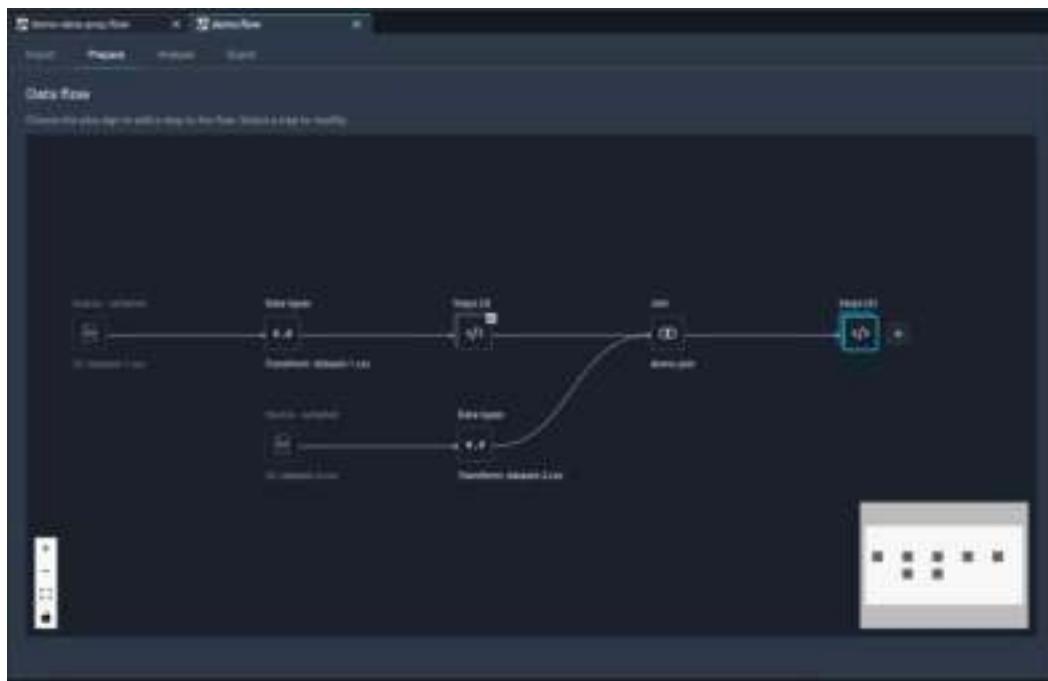
When you import a dataset, the original dataset appears on the data flow and is named **Source**. If you enabled sampling when you imported your data, this dataset is named **Source - sampled**. Data Wrangler automatically infers the types of each column in your dataset and creates a new dataframe named **Data types**. You can select this frame to update the inferred data types. You will see results similar to those shown in the following image after you upload a single dataset:



Each time you add a transform step, you create a new dataframe. When multiple transform steps (other than Join or Concatenate) are added to the same dataset, they are stacked.

Join and Concatenate create stand-alone steps that contain the new joined or concatenated dataset.

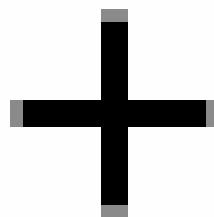
The following diagram shows a data flow with a join between two datasets, as well as two stacks of steps. The first stack (**Steps (2)**) adds two transforms to the type inferred in the **Data types** dataset. The *downstream* stack, or the stack to the right, adds transforms to the dataset resulting from a join named **demo-join**.



The small, gray box in the bottom-right corner of the data flow provides an overview of number of stacks and steps in the flow and the layout of the flow. The lighter box inside the gray box indicates the steps that are within the UI view. You can use this box to see sections of your data flow that fall outside

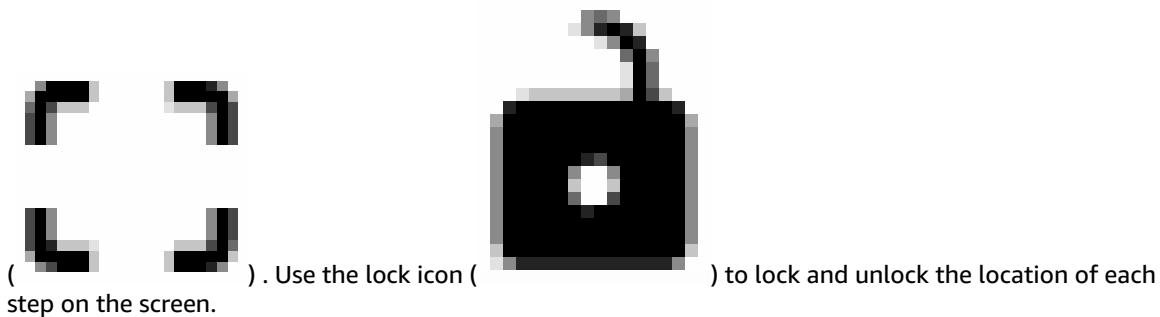


of the UI view. Use the fit screen icon (  ) to fill all steps and datasets into your UI view.



The bottom left navigation bar includes icons that you can use to zoom in (  )

and out (  ) of your data flow and resize the data flow to fit the screen



(). Use the lock icon ( ) to lock and unlock the location of each step on the screen.

## Add a Step to Your Data Flow

Select + next to any dataset or previously added step and then select one of the following options:

- **Edit data types** (For Data types step only): If you have not added any transforms to a **Data types** step, you can select **Edit data types** to update the data types Data Wrangler inferred when importing your dataset.
- **Add transform**: Adds a new transform step. See [Transform Data \(p. 621\)](#) to learn more about the data transformations you can add.
- **Add analysis**: Adds an analysis. You can use this option to analyze your data at any point in the data flow. When you add one or more analyses to a step, an analysis icon ( ) appears on that step. See [Analyze and Visualize \(p. 638\)](#) to learn more about the analyses you can add.
- **Join**: Joins two datasets and adds the resulting dataset to the data flow. To learn more, see [Join Datasets \(p. 626\)](#).
- **Concatenate**: Concatenates two datasets and adds the resulting dataset to the data flow. To learn more, see [Concatenate Datasets \(p. 626\)](#).

## Delete a step from your Data Flow

To delete a step, select the step and select **Delete**. When you delete a step, all subsequent steps connected to that step, or *downstream steps*, are also deleted.

To delete a step from a stack of steps, select the stack and then select the step you want to delete.

### Note

You cannot delete the **Data type** step directly. To remove this dataset, you must delete the corresponding **Source** dataset.

## Transform Data

Amazon SageMaker Data Wrangler provides numerous ML data transforms to streamline cleaning, transforming, and featurizing your data. When you add a transform, it adds a step to the data flow. Each transform you add modifies your dataset and produces a new dataframe. All subsequent transforms apply to the resulting dataframe.

Data Wrangler includes built-in transforms, which you can use to transform columns without any code. You can also add custom transformations using PySpark, Pandas, and PySpark SQL. Some transforms operate in place, while others create a new output column in your dataset.

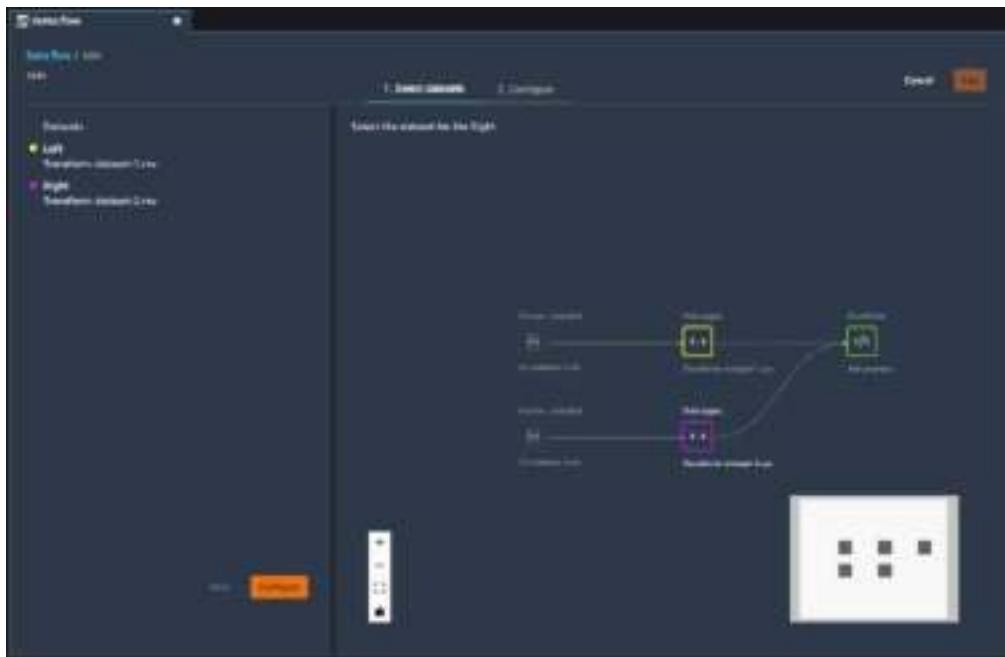
Use this page to learn more about these built-in and custom transforms.

## Transform UI

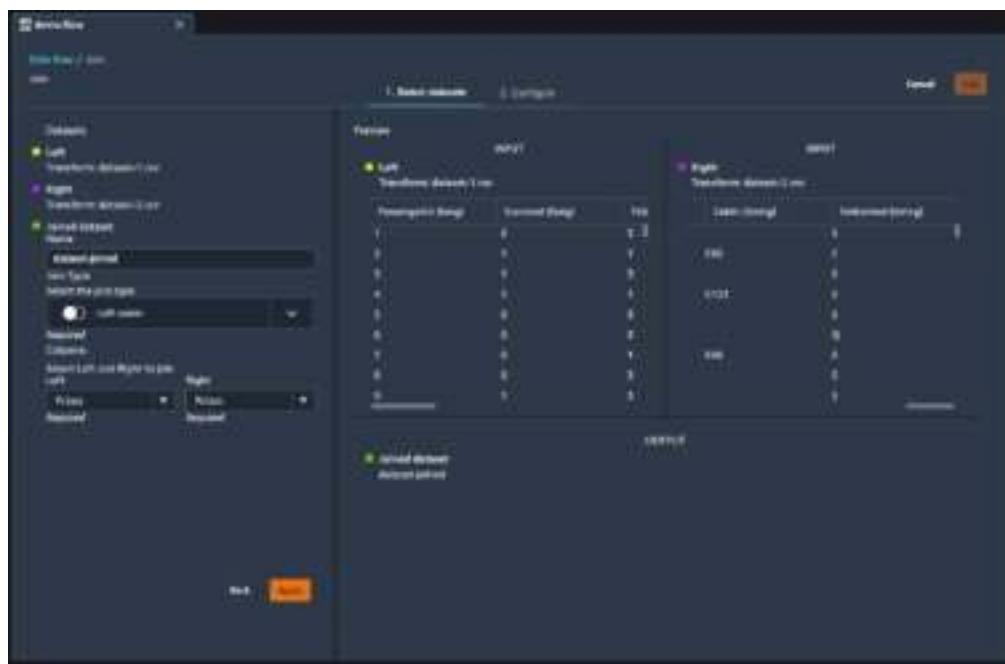
Most of the built-in transforms are located in the **Prepare** tab of the Data Wrangler UI. The Join and Concatenate transforms are accessed through the data flow view. Use the following table to preview these two views.

### Join View

To join two datasets, select the first dataset in your data flow and choose **Join**. When you select **Join**, you see results similar to those shown in the following image. Your left and right datasets are displayed in the left panel. The main panel displays your data flow, with the newly joined dataset added.



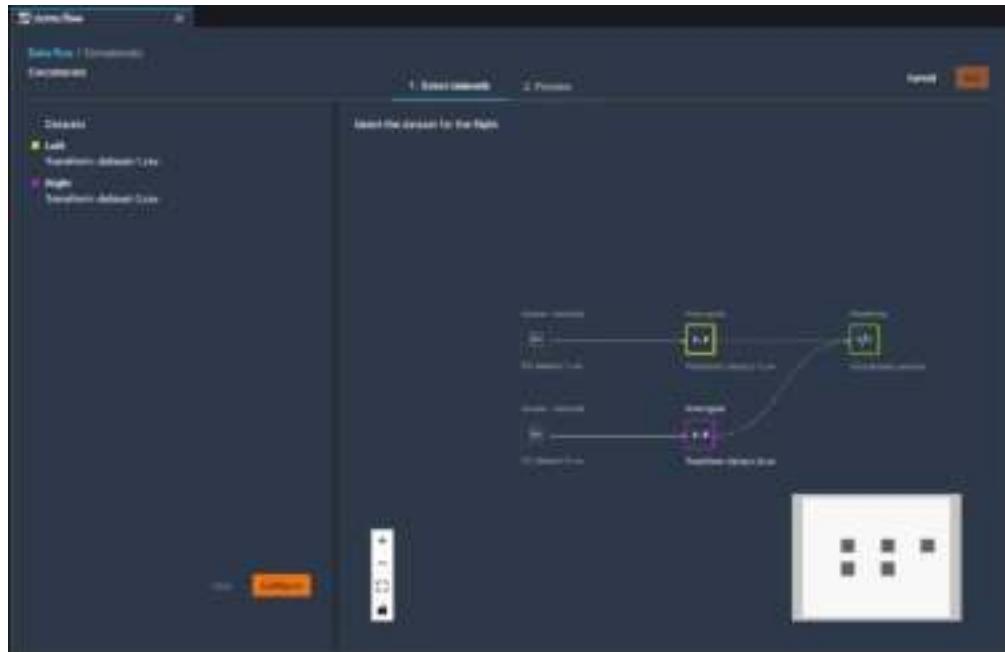
When you select **Configure** to configure your join, you see results similar to those shown in the following image. Your join configuration is displayed in the left panel. You can use this panel to choose the joined dataset name, join type, and columns to join on. The main panel displays three tables. The top-two tables display the Left and Right datasets on the left and right respectively. Under this table, you can preview the joined dataset.



See [Join Datasets \(p. 626\)](#) to learn more.

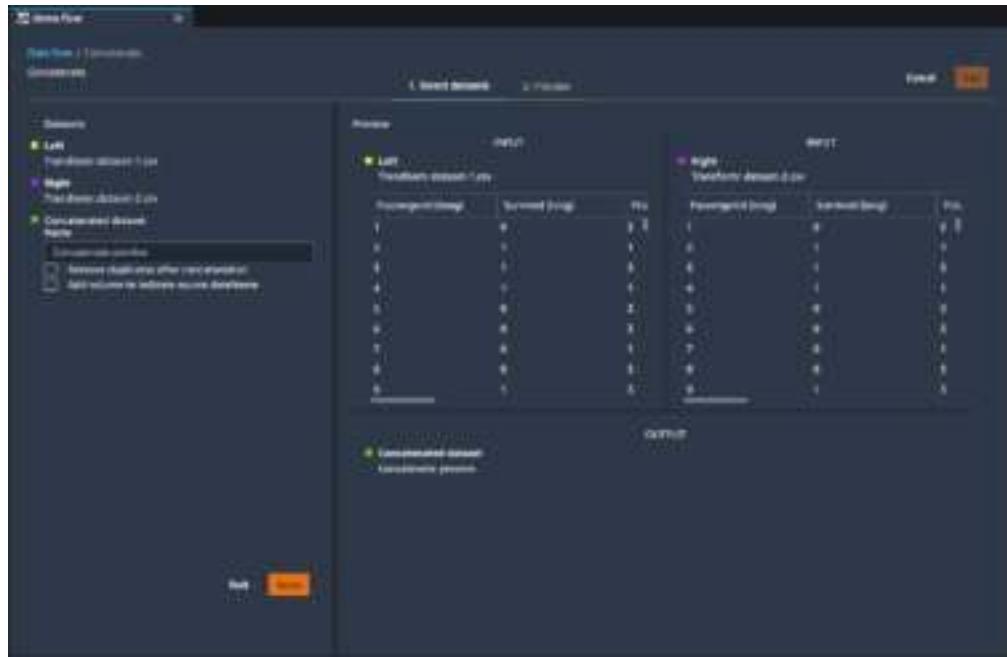
#### Concatenate View

To concatenate two datasets, you select the first dataset in your data flow and choose **Concatenate**. When you select **Concatenate**, you see results similar to those shown in the following image. Your left and right datasets are displayed in the left panel. The main panel displays your data flow, with the newly concatenated dataset added.



When you select **Configure** to configure your concatenation, you see results similar to those shown in the following image. Your concatenate configuration is displayed in the left panel. You can use this panel to choose the concatenated dataset's name, and choose to remove duplicates after

concatenation and add columns to indicate the source dataframe. The main panel displays three tables. The top-two tables display the Left and Right datasets on the left and right respectively. Under this table, you can preview the concatenated dataset.



See [Concatenate Datasets \(p. 626\)](#) to learn more.

#### Transforms in the Prepare Tab

To access transforms on the **Prepare** tab, select **+** next to a step in your data flow, and select **Add transform**.

On the **Prepare** tab, you add steps under **Add**.

The screenshot shows the Amazon SageMaker Data Transformation interface. On the left, there is a table titled "Data File / Transform: titanic-train.csv" containing passenger data. The columns are "PassengerId (String)", "Survived (String)", "Pclass (String)", and "Name (String)". The table has 41 rows of data. On the right, there is a sidebar titled "Transformation" with a list of available transformations:

- 1. Back to data flow
- 2. Gaussian Transform
- 3. Custom formula
- 4. Divide/integer
- 5. Evaluate Text
- 6. Extract substrings
- 7. Format String
- 8. Handle outliers
- 9. Handling missing values
- 10. Merge columns
- 11. Merge rows
- 12. Scale columns
- 13. Search and edit
- 14. Type Conversion
- 15. Validate rows

You can use the **Previous steps** tab to view and remove transformations that you have added, in sequential order.

The screenshot shows the Amazon SageMaker Data Transformation interface with the "Previous steps" tab selected. The table on the left is the same as the previous screenshot. The right sidebar shows the "Transformation" section with the "Previous steps" tab highlighted. It lists the following steps:

- 1. Back to data flow
- 2. Map applied to this stepname
- 3. 1 - 1. S3 Source
- 4. 1 - 2. Data types

## Join Datasets

You join dataframes directly in your data flow. When you join two datasets, the resulting joined dataset appears in your flow. The following join types are supported by Data Wrangler.

- **Left Outer** – Include all rows from the left table. If the value for the column joined on in a left table row does not match any right table row values, that row contains null values for all right table columns in the joined table.
- **Left Anti** – Include rows from the left table that do not contain values in the right table for the joined column.
- **Left semi** – Include a single row from the left table for all identical rows that satisfy the criteria in the join statement. This excludes duplicate rows from the left table that match the criteria of the join.
- **Right Outer** – Include all rows from the right table. If the value for the joined column in a right table row does not match any left table row values, that row contains null values for all left table columns in the joined table.
- **Inner** – Include rows from left and right tables that contain matching values in the joined column.
- **Full Outer** – Include all rows from the left and right tables. If the row value for the joined column in either table does not match, separate rows are created in the joined table. If a row doesn't contain a value for a column in the joined table, null is inserted for that column.
- **Cartesian Cross** – Include rows which combine each row from the first table with each row from the second table. This is a [Cartesian product](#) of rows from tables in the join. The result of this product is the size of the left table times the size of the right table. Therefore, we recommend caution in using this join between very large datasets.

Use the following procedure to join two dataframes.

1. Select + next to the left dataframe that you want to join. The first dataframe you select is always the left table in your join.
2. Select **Join**.
3. Select the right dataframe. The second dataframe you select is always the right table in your join.
4. Select **Configure** to configure your join.
5. Give your joined dataset a name using the **Name** field.
6. Select a **Join type**.
7. Select a column from the left and right tables to join on.
8. Select **Apply** to preview the joined dataset on the right.
9. To add the joined table to your data flow, select **Add** in the top right corner.

## Concatenate Datasets

### Concatenate two datasets:

1. Select + next to the left dataframe that you want to concatenate. The first dataframe you select is always the left table in your concatenate.
2. Select **Concatenate**.
3. Select the right dataframe. The second dataframe you select is always the right table in your concatenate.
4. Select **Configure** to configure your concatenate.
5. Give your concatenated dataset a name using the **Name** field.
6. (Optional) Select the check box next to **Remove duplicates after concatenation** to remove duplicate columns.

7. (Optional) Select the check box next to **Add column to indicate source dataframe** if, for each column in the new dataset, you want to add an indicator of the column's source.
8. Select **Apply** to preview the new dataset.
9. Select **Add** to add the new dataset to your data flow.

## Custom Transforms

The **Custom Transforms** group allows you to use Pyspark, Pandas, or Pyspark (SQL) to define custom transformations. For all three options, you use the variable `df` to access the dataframe to which you want to apply the transform. You do not need to include a return statement. Select **Preview** to preview the result of the custom transform. Select **Add** to add the custom transform to your list of **Previous steps**.

You can import the popular libraries with an `import` statement in the custom transform code block such as the following:

- Numpy version 1.19.0
- Scikit-learn version 0.23.2
- Scipy version 1.5.4
- Pandas version 1.0.3
- Pyspark version 3.0.0

If you include print statements in the code block, the result appears when you select **Preview**.

The following are examples of how you can use the custom transforms code block:

### Pyspark

The following example extracts date and time from a timestamp.

```
from pyspark.sql.functions import from_unixtime, to_date, date_format
df = df.withColumn('DATE_TIME', from_unixtime('TIMESTAMP'))
df = df.withColumn( 'EVENT_DATE', to_date('DATE_TIME')).withColumn(
'EVENT_TIME', date_format('DATE_TIME', 'HH:mm:ss'))
```

### Pandas

The following example provides an overview of the dataframe to which you are adding transforms.

```
df.info()
```

### Pyspark (SQL)

The following creates a new dataframe with five columns: `name`, `fare`, `pclass`, `survived`.

```
SELECT name, fare, pclass, survived FROM df
```

## Custom Formula

Use **Custom formula** to define a new column using a Spark SQL expression to query data in the current dataframe. The query must use the conventions of Spark SQL expressions.

### Important

**Custom formula** does not support columns with spaces in the name. You can use the **Rename column** transform in the **Manage columns** transform group to remove spaces from a column's name. You can also add a **Pandas Custom transform** similar to the following to remove spaces from multiple columns in a single step. This example changes columns named **A column** and **B column** to **A\_column** and **B\_column** respectively.

```
df.rename(columns={"A column": "A_column", "B column": "B_column"})
```

You can use this transform to perform operations on columns, referencing the columns by name. For example, assuming the current dataframe contains columns named **col\_a** and **col\_b**, you can use the following operation to produce an **Output column** that is the product of these two columns with the following code:

```
col_a * col_b
```

Other common operations include the following, assuming a dataframe contains **col\_a** and **col\_b** columns:

- Concatenate two columns: `concat(col_a, col_b)`
- Add two columns: `col_a + col_b`
- Subtract two columns: `col_a - col_b`
- Divide two columns: `col_a / col_b`
- Take the absolute value of a column: `abs(col_a)`

For more information, see the [Spark documentation](#) on selecting data.

## Encode Categorical

Categorical data is usually composed of a finite number of categories, where each category is represented with a string. For example, if you have a table of customer data, a column that indicates the country a person lives in is categorical. The categories would be Afghanistan, Albania, Algeria, and so on. Categorical data can be *nominal* or *ordinal*. Ordinal categories have an inherent order, and nominal categories do not. The highest degree obtained (High school, Bachelors, Master) is an example of ordinal categories.

Encoding categorical data is the process of creating a numerical representation for categories. For example, if your categories are Dog and Cat, you may encode this information into two vectors, [1, 0] to represent dog, and [0, 1] to represent cat.

When you encode ordinal categories, you may need to translate the natural order of categories into your encoding. For example, you can represent highest degree obtained with the following map: {"High school": 1, "Bachelors": 2, "Masters": 3}.

Use categorical encoding to encode categorical data that is in string format into arrays of integers.

The Data Wrangler categorical encoders create encodings for all categories that exist in a column at the time the step is defined. If new categories have been added to a column when you start a Data Wrangler job to process your dataset at time  $t$ , and this column was the input for a Data Wrangler categorical encoding transform at time  $t-1$ , these new categories are considered *missing* in the Data Wrangler job. The option you select for **Invalid handling strategy** is applied to these missing values. Examples of when this can occur are:

- When you use a .flow file to create a Data Wrangler job to process a dataset that was updated after the creation of the data flow. For example, you may use a data flow to regularly process sales data each

month. If that sales data is updated weekly, new categories may be introduced into columns for which an encode categorical step is defined.

- When you select **Sampling** when you import your dataset, some categories may be left out of the sample.

In these situations, these new categories are considered missing values in the Data Wrangler job.

You can choose from and configure an *ordinal* and a *one-hot encode*. Use the following sections to learn more about these options.

Both transforms create a new column named **Output column name**. You specify the output format of this column with **Output style**:

- Select **Vector** to produce a single column with a sparse vector.
- Select **Columns** to create a column for every category with an indicator variable for whether the text in the original column contains a value that is equal to that category.

## Ordinal Encode

Select **Ordinal encode** to encode categories into an integer between 0 and the total number of categories in the **Input column** you select.

**Invalid handing strategy:** Select a method to handle invalid or missing values.

- Choose **Skip** if you want to omit the rows with missing values.
- Choose **Keep** to retain missing values as the last category.
- Choose **Error** if you want Data Wrangler to throw an error if missing values are encountered in the **Input column**.
- Choose **Replace with NaN** to replace missing with NaN. This option is recommended if your ML algorithm can handle missing values. Otherwise, the first three options in this list may produce better results.

## One-Hot Encode

Select **One-hot encode** for **Transform** to use one-hot encoding. Configure this transform using the following:

- **Drop last category:** If **True**, the last category does not have a corresponding index in the one-hot encoding. When missing values are possible, a missing category is always the last one and setting this to **True** means that a missing value results in an all zero vector.
- **Invalid handing strategy:** Select a method to handle invalid or missing values.
  - Choose **Skip** if you want to omit the rows with missing values.
  - Choose **Keep** to retain missing values as the last category.
  - Choose **Error** if you want Data Wrangler to throw an error if missing values are encountered in the **Input column**.
- **Is input ordinal encoded:** Select this option if the input vector contains ordinal encoded data. This option requires that input data contain non-negative integers. If **True**, input  $i$  is encoded as a vector with a non-zero in the  $i$ th location.

## Featurize Text

Use the **Feature Text** transform group to inspect string typed columns and use text embedding to featurize these columns.

This feature group contains two features, *Character statistics* and *Vectorize*. Use the following sections to learn more about these transforms. For both options, the **Input column** must contain text data (string type).

## Character Statistics

Use **Character statistics** to generate statistics for each row in a column containing text data.

This transform computes the following ratios and counts for each row, and creates a new column to report the result. The new column is named using the input column name as a prefix and a suffix that is specific to the ratio or count.

- Number of words: The total number of words in that row. The suffix for this output column is **-stats\_word\_count**.
- Number of characters: The total number of characters in that row. The suffix for this output column is **-stats\_char\_count**.
- Ratio of upper: The number of upper-case characters, from A to Z, divided by all characters in the column. The suffix for this output column is **-stats\_capital\_ratio**.
- Ratio of lower: The number of lower-case characters, from a to z, divided by all characters in the column. The suffix for this output column is **-stats\_lower\_ratio**.
- Ratio of digits: The ratio of digits in a single row over the sum of digits in the input column. The suffix for this output column is **-stats\_digit\_ratio**.
- Special characters ratio: The ratio of non-alphanumeric (characters like #&@) characters to over the sum of all characters in the input column. The suffix for this output column is **-stats\_special\_ratio**.

## Vectorize

Text embedding involves mapping words or phrases from a vocabulary to vectors of real numbers. Use the Data Wrangler text embedding transform to tokenize and vectorize text data into term frequency-inverse document frequency (TF-IDF) vectors.

When TF-IDF is calculated for a column of text data, each word in each sentence is converted to a real number that represents its semantic importance. Higher numbers are associated with less frequent words, which tend to be more meaningful.

When you define a **Vectorize** transform step, the count vectorizer and TF-IDF methods are defined using data available in Data Wrangler when defining this step. These same methods will be used when running a Data Wrangler job.

You configure this transform using the following:

- **Output column name:** This transform will create a new column with the text embedding. Use this field to specify a name for this output column.
- **Tokenizer:** A tokenizer converts the sentence into a list of words, or *tokens*.

Choose **Standard** to use a tokenizer that splits by white space and converts each word to lowercase. For example, "Good dog" is tokenized to ["good", "dog"].

Choose **Custom** to use a customized tokenizer. If you choose **Custom**, you can use the following fields to configure the tokenizer:

- **Minimum token length:** The minimum length, in characters, for a token to be valid. Defaults to 1. For example, if you specify 3 for minimum token length, words like a, at, in are dropped from the tokenized sentence.
- **Should regex split on gaps:** If selected, **regex** splits on gaps. Otherwise, it matches tokens. Defaults to True.
- **Regex pattern:** Regex pattern that defines the tokenization process. Defaults to '\s+'.

- **To lowercase:** If chosen, all characters are converted to lowercase before tokenization. Defaults to `True`.

To learn more, refer to the Spark documentation on [Tokenizer](#).

- **Vectorizer:** The vectorizer converts the list of tokens into a sparse numeric vector. Each token corresponds to an index in the vector and a non-zero indicates the existence of the token in the input sentence. You can choose from two vectorizer options, `Count` and `Hashing`.
  - **Count vectorize** allows customizations that filter infrequent or too common tokens. **Count vectorize parameters** include the following:
    - **Minimum term frequency:** In each row, terms (tokens) with smaller frequency are filtered. If you specify an integer, this is an absolute threshold (inclusive). If you specify a fraction between 0 (inclusive) and 1, the threshold is relative to the total term count. Defaults to 1.
    - **Minimum document frequency:** Minimum number of rows in which a term (token) must appear to be included. If you specify an integer, this is an absolute threshold (inclusive). If you specify a fraction between 0 (inclusive) and 1, the threshold is relative to the total term count. Defaults to 0.1.
    - **Maximum document frequency:** Maximum number of documents (rows) in which a term (token) can appear to be included. If you specify an integer, this is an absolute threshold (inclusive). If you specify a fraction between 0 (inclusive) and 1, the threshold is relative to the total term count. Defaults to 0.999.
    - **Maximum vocabulary size:** Maximum size of the vocabulary. The vocabulary is made up of all terms (tokens) in all rows of the column. Defaults to 262144.
    - **Binary outputs:** If selected, the vector outputs do not include the number of appearances of a term in a document, but rather are a binary indicator of its appearance. Defaults to `False`.

To learn more about this option, refer to the Spark documentation on [CountVectorizer](#).

- **Hashing** is computationally faster. **Hash vectorize parameters** includes the following:
  - **Number of features during hashing:** A hash vectorizer maps tokens to a vector index according to their hash value. This feature determines the number of possible hash values. Large values result in fewer collisions between hash values but a higher dimension output vector.

To learn more about this option, refer to the Spark documentation on [FeatureHasher](#)

- **Apply IDF:** When chosen, an IDF transformation is applied, which multiplies the term frequency with the standard inverse document frequency used for TF-IDF embedding. **IDF parameters** include the following:
  - **Minimum document frequency :** Minimum number of documents (rows) in which a term (token) must appear to be included. If `count_vectorize` is the chosen vectorizer, we recommend that you keep the default value and only modify the `min_doc_freq` field in **Count vectorize parameters**. Defaults to 5.
  - **Output format:** The output format of each row.
    - Select **Vector** to produce a single column with a sparse vector.
    - Select **Flattened** to create a column for every category with an indicator variable for whether the text in the original column contains a value that is equal to that category. You can only choose flattened when `Vectorizer` is set as `Count vectorizer`.

## Featurize Date/Time

Use **Featurize date/time** to create a vector embedding representing a date/time field. To use this transform, your date/time data must be in one of the following formats:

- Strings describing date/time, for example, "January 1st, 2020, 12:44pm".
- A unix timestamp. A unix timestamp describes the number of seconds, milliseconds, microseconds, or nanoseconds from 1/1/1970.

You can choose to **Infer datetime format** and provide a **Datetime format**. If you provide a date/time format, you must use the codes described in this [Python documentation](#). The options you select for these two configurations have implications for the speed of the operation, and the final results:

- The most manual and computationally fastest option is to specify a **Datetime format** and select **No** for **Infer datetime format**.
- To reduce manual labor, you can simply choose **Infer datetime format** and not specify a date/time format. This is also a computationally fast operation; however, the first date/time format encountered in the input column is assumed to be the format for the entire column. Therefore, if other formats are encountered in the column, these values are NaN in the final output. Therefore, this option can result in unparsed strings.
- If you do not specify a format and you select **No** for **Infer datetime format**, you get the most robust results. All valid date/time strings are parsed. However, this operation can be an order of magnitude slower than the first two options in this list.

When you use this transform, you specify an **Input column** which contains date/time data in one of the formats listed above. The transform creates an output column named **Output column name**. The format of the output column depends on your configuration using the following:

- **Vector**: Outputs a single column with a sparse vector.
- **Columns**: Creates a new column for every feature. For example, if the output contains a year, month, and day, three separate columns are created for year, month, and day.

Additionally, you must choose an **Embedding mode**. For linear models and deep networks, **cyclic** is recommended. For tree based algorithms, **ordinal** is recommended.

## Format String

The **Format string** transforms contain standard string formatting operations. For example, you can use these operations to remove special characters, normalize string lengths, and update string casing.

This feature group contains the following transforms. All transforms return copies of the strings in the **Input column** and add the result to a new, output column.

Name	Function
Left pad	Left-pad the string with a given <b>Fill character</b> to the given <b>width</b> . If the string is longer than <b>width</b> , the return value is shortened to <b>width</b> characters.
Right pad	Right-pad the string with a given <b>Fill character</b> to the given <b>width</b> . If the string is longer than <b>width</b> , the return value is shortened to <b>width</b> characters.
Center (pad on either side)	Center-pad the string (add padding on both sides of the string) with a given <b>Fill character</b> to the given <b>width</b> . If the string is longer than <b>width</b> , the return value is shortened to <b>width</b> characters.
Prepend zeros	Left-fill a numeric string with zeros, up to a given <b>width</b> . If the string is longer than <b>width</b> , the return value is shortened to <b>width</b> characters.
Strip left and right	Returns a copy of the string with the leading and trailing characters removed.

Name	Function
Strip characters from left	Returns a copy of the string with leading characters removed.
Strip characters from right	Returns a copy of the string with trailing characters removed.
Lower case	Convert all letters in text to lowercase.
Upper case	Convert all letters in text to uppercase.
Capitalize	Capitalize the first letter in each sentence.
Swap case	Converts all uppercase characters to lowercase and all lowercase characters to uppercase characters of the given string, and returns it.
Add prefix or suffix	Adds a prefix and a suffix to the string column. You must specify at least one of <b>Prefix</b> and <b>Suffix</b> .
Remove Symbols	Removes given symbols from a string. All listed characters will be removed. Defaults to white space.

## Handle Outliers

Machine learning models are sensitive to the distribution and range of your feature values. Outliers, or rare values, can negatively impact model accuracy and lead to longer training times. Use this feature group to detect and update outliers in your dataset.

When you define a **Handle outliers** transform step, the statistics used to detect outliers are generated on the data available in Data Wrangler when defining this step. These same statistics are used when running a Data Wrangler job.

Use the following sections to learn more about the transforms this group contains. You specify an **Output name** and each of these transforms produces an output column with the resulting data.

### Robust standard deviation numeric outliers

This transform detects and fixes outliers in numeric features using statistics that are robust to outliers.

You must define an **Upper quantile** and a **Lower quantile**, which are used in the statistics used to calculate outliers. You also need to specify the number of **Standard deviations** from which a value must vary from the mean to be considered an outlier. For example, if you specify 3 for **Standard deviations**, a value must fall more than 3 standard deviations from the mean to be considered an outlier.

The **Fix method** is the method used to handle outliers when they are detected. You can choose from the following:

- **Clip:** Use this option to clip the outliers to the corresponding outlier detection bound.
- **Remove:** Use this to remove rows with outliers from the dataframe.
- **Invalidate:** Use this to replace outliers with invalid values.

### Standard Deviation Numeric Outliers

This transform detects and fixes outliers in numeric features using the mean and standard deviation.

You specify the number of **Standard deviations** a value must vary from the mean to be considered an outlier. For example, if you specify 3 for **Standard deviations**, a value must fall more than 3 standard deviations from the mean to be considered an outlier.

The **Fix method** is the method used to handle outliers when they are detected. You can choose from the following:

- **Clip**: Use this option to clip the outliers to the corresponding outlier detection bound.
- **Remove**: Use this to remove rows with outliers from the dataframe.
- **Invalidate**: Use this to replace outliers with invalid values.

## Quantile Numeric Outliers

Use this transform to detect and fix outliers in numeric features using quantiles. You can define an **Upper quantile** and a **Lower quantile**, and all values that fall above or below those quantile-values, respectively, are considered outliers.

The **Fix method** is the method used to handle outliers when they are detected. You can choose from the following:

- **Clip**: Use this option to clip the outliers to the corresponding outlier detection bound.
- **Remove**: Use this to remove rows with outliers from the dataframe.
- **Invalidate**: Use this to replace outliers with invalid values.

## Min-Max Numeric Outliers

This transform detects and fixes outliers in numeric features using upper and lower thresholds. Use this method if you know threshold values that demark outliers.

You specify a **Upper threshold** and a **Lower threshold**, and if values fall above or below those thresholds respectively, they are considered outliers.

The **Fix method** is the method used to handle outliers when they are detected. You can choose from the following:

- **Clip**: Use this option to clip the outliers to the corresponding outlier detection bound.
- **Remove**: Use this to remove rows with outliers from the dataframe.
- **Invalidate**: Use this to replace outliers with invalid values.

## Replace Rare

When you use the **Replace rare** transform, you specify a threshold and Data Wrangler finds all values that meet that threshold and replaces them with a string that you specify. For example, you may want to use this transform to categorize all outliers in a column into an "Others" category.

- **Replacement string**: The string with which to replace outliers.
- **Absolute threshold**: A category is rare if the number of instances is less than or equal to this absolute threshold.
- **Fraction threshold**: A category is rare if the number of instances is less than or equal to this fraction threshold multiplied by the number of rows.
- **Max common categories**: Maximum not-rare categories that remain after the operation. If the threshold does not filter enough categories, those with the top number of appearances are classified as not rare. If set to 0 (default), there is no hard limit to the number of categories.

## Handle Missing Values

Missing values are a common occurrence in machine learning datasets. In some situations, it is appropriate to impute missing data with a calculated value, such as an average or categorically common value. You can process missing values using the **Handle missing values** transform group. This group contains the following transforms.

### Fill Missing

Use the **Fill missing** transform to replace missing values with a **Fill value** you define.

### Impute Missing

Use the **Impute missing** transform to create a new column that contains imputed values where missing values were found in input categorical and numerical data. The configuration depends on your data type. Configure this transform using the following:

- **Imputing strategy:** The strategy used to determine the new value to impute.

For numeric data, the chosen statistic is computed over the present values and is used as the imputed value for all missing values. The choices are **mean** and **median**.

For categorical data, you can choose to impute the **Most frequent value** in the column, or you can define a custom string to impute.

### Add Indicator for Missing

Use the **Add indicator for missing** transform to create a new indicator column, which contains a boolean "false" if a row contains a value, and "true" if a row contains a missing value.

### Drop Missing

Use the **Drop missing** option to drop rows that contain missing values from the **Input column**.

## Manage Columns

You can use the following transforms to quickly update and manage columns in your dataset:

Name	Function
Drop Column	Delete a column.
Duplicate Column	Duplicate a column.
Rename Column	Rename a column.
Move Column	Move a column's location in the dataset. Choose to move your column to the start or end of the dataset, before or after a reference column, or to a specific index.

## Manage Rows

Use this transform group to quickly perform sort and shuffle operations on rows. This group contains the following:

- **Sort:** Sort the entire dataframe by a given column. Select the check box next to **Ascending order** for this option; otherwise, deselect the check box and descending order is used for the sort.
- **Shuffle:** Randomly shuffle all rows in the dataset.

## Manage Vectors

Use this transform group to combine or flatten vector columns. This group contains the following transforms.

- **Assemble:** Use this transform to combine Spark vectors and numeric data into a single column. For example, you can combine three columns: two containing numeric data and one containing vectors. Add all the columns you want to combine in **Input columns** and specify a **Output column name** for the combined data.
- **Flatten:** Use this transform to flatten a single column containing vector data. The input column must contain PySpark vectors, or array-like objects. You can control the number of columns that get created by specifying a **Method to detect number of outputs**. For example, if you select **Length of first vector**, the number of elements in the first valid vector or array found in the column determines the number of output columns that gets created. All other input vectors with too many items are truncated. Inputs with too few items are filled with NaNs.

You also specify an **Output prefix**, which is used as the prefix for each output column.

## Process Numeric

Use the **Process Numeric** feature group to process numeric data. Each scalar in this group is defined using the Spark library. The following scalars are supported:

- **Standard Scaler:** Standardize the input column by subtracting the mean from each value and scaling to unit variance. To learn more, refer to the Spark documentation for [StandardScaler](#).
- **Robust Scaler:** Scale the input column using statistics that are robust to outliers. To learn more, refer to the Spark documentation for [RobustScaler](#).
- **Min Max Scaler:** Transform the input column by scaling each feature to a given range. To learn more, refer to the Spark documentation for [MinMaxScaler](#).
- **Max Absolute Scaler:** Scale the input column by dividing each value by the maximum absolute value. To learn more, refer to the Spark documentation for [MaxAbsScaler](#).

## Search and Edit

Use this section to search for and edit specific patterns within strings. For example, you can find and update strings within sentences or documents, split strings by delimiters, and find occurrences of specific strings.

The following transforms are supported under **Search and edit**. All transforms return copies of the strings in the **Input column** and add the result to a new output column.

Name	Function
Find substring	Returns the index of the first occurrence of the <b>Substring</b> for which you searched optionally, starting and ending the search at <b>Start</b> and <b>End</b> respectively.
Find substring (from right)	Returns the index of the last occurrence of the <b>Substring</b> for which you searched, optionally,

Name	Function
	starting and ending the search at <b>Start</b> and <b>End</b> respectively.
Matches prefix	Returns a Boolean if the string contains a given <b>Pattern</b> . A pattern can be a character sequence or regular expression. Optionally, you can make the pattern case sensitive.
Find all occurrences	Returns an array with all occurrences of a given pattern. A pattern can be a character sequence or regular expression.
Extract using regex	Returns a string that matches a given Regex pattern.
Extract between delimiters	Returns a string with all characters found between <b>Left delimiter</b> and <b>Right delimiter</b> .
Extract from position	Returns a string, starting from <b>Start position</b> in the input string, that contains all characters up to the start position plus <b>Length</b> .
Find and replace substring	Returns a string with all matches of a given <b>Pattern</b> (regular expression) replaced by <b>Replacement string</b> .
Replace between delimiters	Returns a string with the substring found between the first appearance of a <b>Left delimiter</b> and the last appearance of a <b>Right delimiter</b> replaced by <b>Replacement string</b> . If no match is found, nothing is replaced.
Replace from position	Returns a string with the substring between <b>Start position</b> and <b>Start position</b> plus <b>Length</b> replaced by <b>Replacement string</b> . If <b>Start position</b> plus <b>Length</b> is greater than the length of the replacement string, the output contains ....
Convert regex to missing	Converts a string to <b>None</b> if invalid and returns the result. Validity is defined with a regular expression in <b>Pattern</b> .
Split string by delimiter	Returns an array of strings from the input string, split by <b>Delimiter</b> , with up to <b>Max number of splits</b> (optional). Delimiter defaults to white space.

## Parse Value as Type

Use this transform to cast a column to a new type. The supported Data Wrangler data types are:

- Long
- Float
- Boolean
- Date, in the format dd-MM-yyyy, representing day, month, and year respectively.

- String

## Validate String

Use the **Validate string** transforms to create a new column that indicates that a row of text data meets a specified condition. For example, you can use a **Validate string** transform to verify that a string only contains lowercase characters. The following transforms are supported under **Validate string**.

The following transforms are included in this transform group. If a transform outputs a Boolean value, `True` is represented with a 1 and `False` is represented with a 0.

Name	Function
String length	Returns <code>True</code> if a string length equals specified length. Otherwise, returns <code>False</code> .
Starts with	Returns <code>True</code> if a string starts will a specified prefix. Otherwise, returns <code>False</code> .
Ends with	Returns <code>True</code> if a string length equals specified length. Otherwise, returns <code>False</code> .
Is alphanumeric	Returns <code>True</code> if a string only contains numbers and letters. Otherwise, returns <code>False</code> .
Is alpha (letters)	Returns <code>True</code> if a string only contains letters. Otherwise, returns <code>False</code> .
Is digit	Returns <code>True</code> if a string only contains digits. Otherwise, returns <code>False</code> .
Is space	Returns <code>True</code> if a string only contains numbers and letters. Otherwise, returns <code>False</code> .
Is title	Returns <code>True</code> if a string contains any white spaces. Otherwise, returns <code>False</code> .
Is lowercase	Returns <code>True</code> if a string only contains lower case letters. Otherwise, returns <code>False</code> .
Is uppercase	Returns <code>True</code> if a string only contains upper case letters. Otherwise, returns <code>False</code> .
Is numeric	Returns <code>True</code> if a string only contains numbers. Otherwise, returns <code>False</code> .
Is decimal	Returns <code>True</code> if a string only contains decimal numbers. Otherwise, returns <code>False</code> .

## Analyze and Visualize

Amazon SageMaker Data Wrangler includes built-in analyses that help you generate visualizations and data analyses in a few clicks. You can also create custom analyses using your own code.

You add an analysis to a dataframe by selecting a step in your data flow, and then choosing **Add analysis**. To access an analysis you've created, select the step that contains the analysis, and select the analysis.

All analyses are generated using 100,000 rows of your dataset.

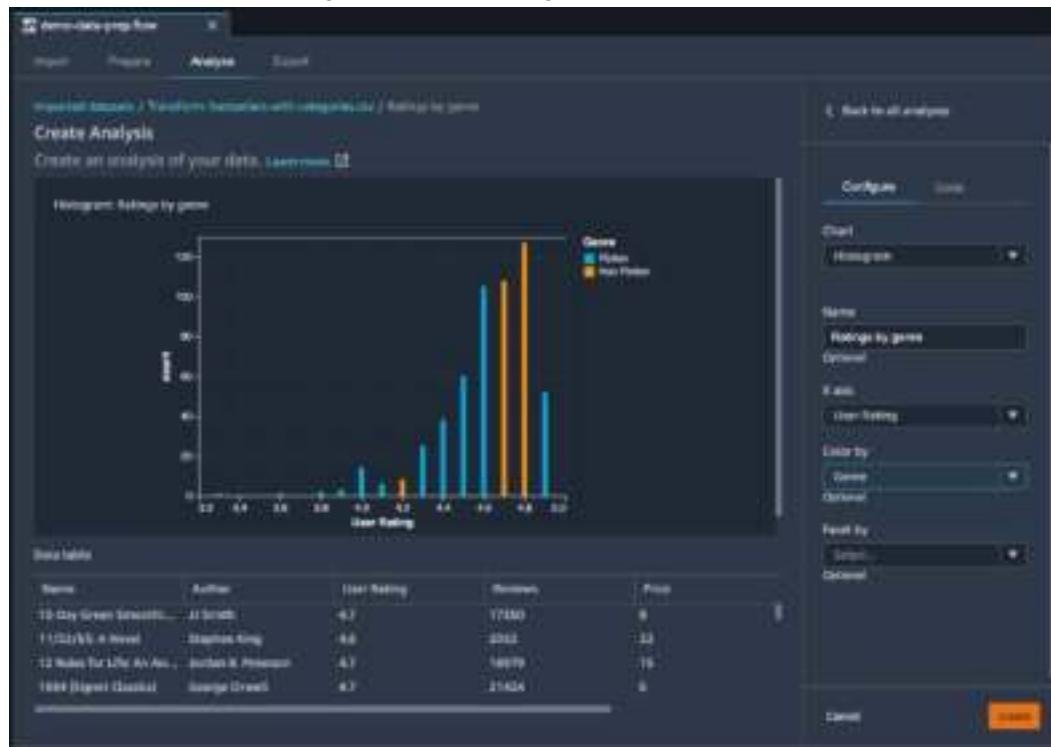
You can add the following analysis to a dataframe:

- Data visualizations, including histograms and scatter plots.
- A quick summary of your dataset, including number of entries, minimum and maximum values (for numeric data), and most and least frequent categories (for categorical data).
- A quick model of the dataset, which can be used to generate an importance score for each feature.
- A target leakage report, which you can use to determine if one or more features are strongly correlated with your target feature.
- A custom visualization using your own code.

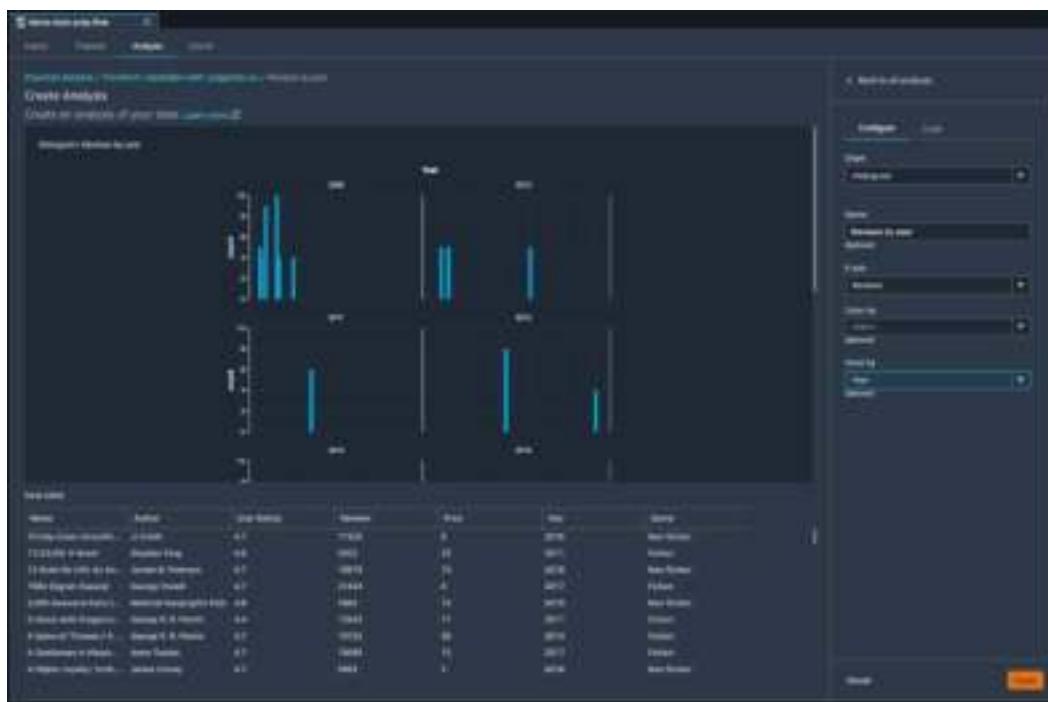
Use the following sections to learn more about these options.

## Histogram

Use histograms to see the counts of feature values for a specific feature. You can inspect the relationships between features using the **Color by** option. For example, the following histogram charts the distribution of user ratings of the best-selling books on Amazon from 2009–2019, colored by genre.



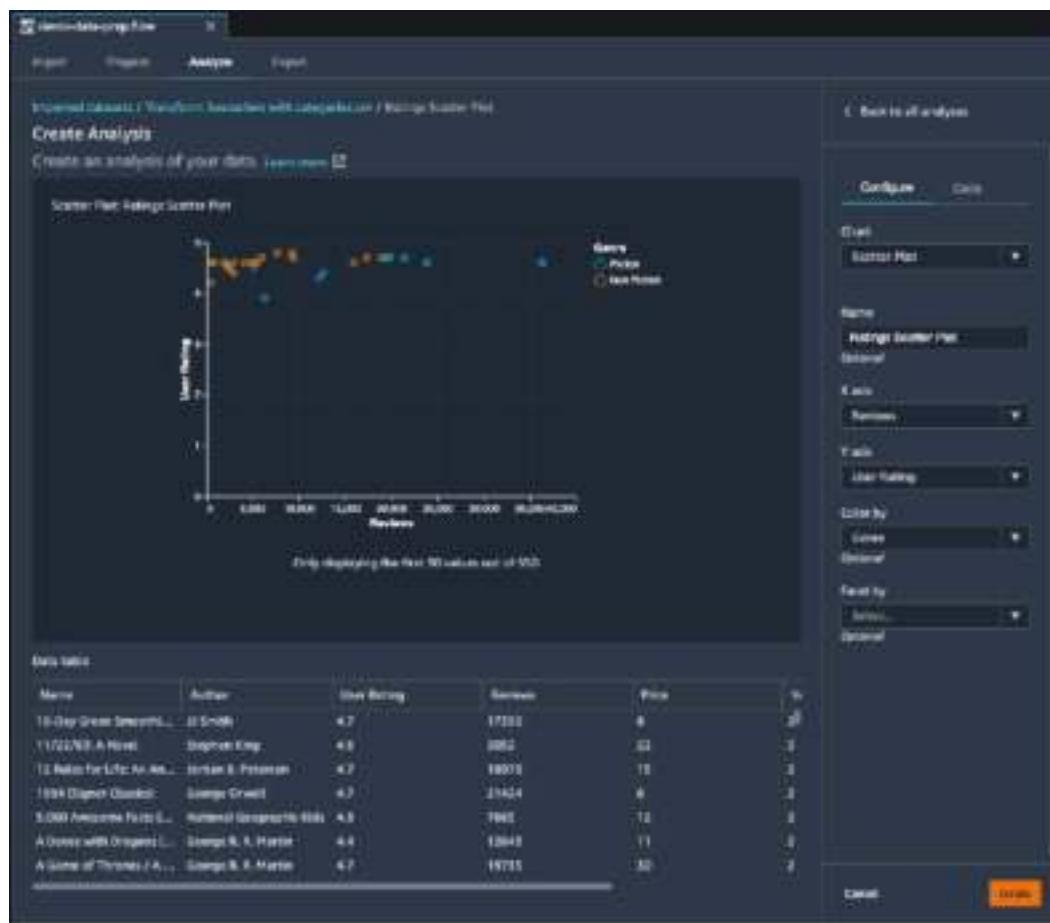
You can use the **Facet by** feature to create histograms of one column, for each value in another column. For example, the following diagram shows histograms of user reviews of best-selling books on Amazon if faceted by year.



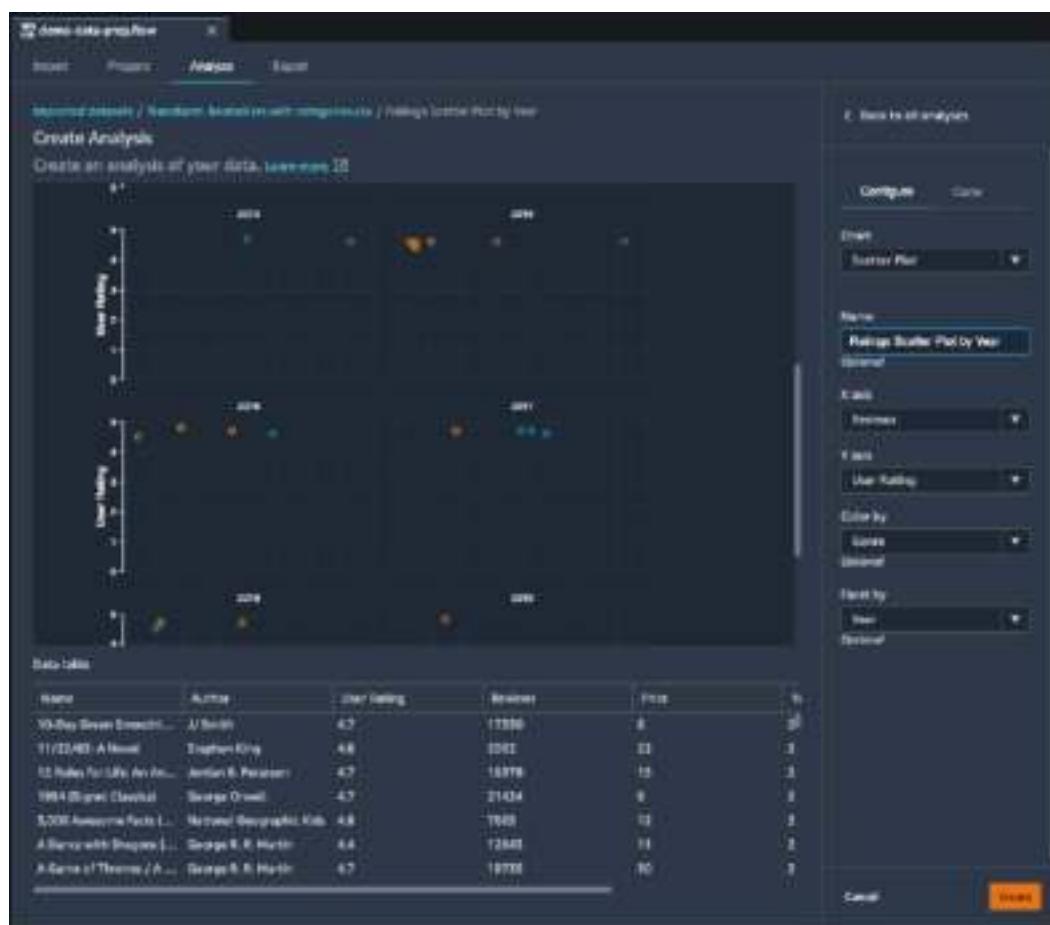
## Scatter Plot

Use the **Scatter Plot** feature to inspect the relationship between features. To create a scatter plot, select a feature to plot on the **X axis** and the **Y axis**. Both of these columns must be numeric typed columns.

You can color scatter plots by an additional column. For example, the following example shows a scatter plot comparing the number of reviews against user ratings of top-selling books on Amazon between 2009 and 2019. The scatter plot is colored by book genre.



Additionally, you can facet scatter plots by features. For example, the following shows an example of the same review versus user rating scatter plot, faceted by year.



## Table Summary

Use the **Table Summary** analysis to quickly summarize your data.

For columns with numerical data, including log and float data, a table summary reports the number of entries (count), minimum (min), maximum (max), mean, and standard deviation (stddev) for each column.

For columns with non-numerical data, including columns with string, Boolean, or date/time data, a table summary reports the number of entries (count), least frequent value (min), and most frequent value (max).

## Quick Model

Use the **Quick Model** visualization to quickly evaluate your data and produce importance scores for each feature. A [feature importance score](#) score indicates how useful a feature is at predicting a target label. The feature importance score is between [0, 1] and a higher number indicates that the feature is more important to the whole dataset. On the top of the quick model chart, there is a model score. A classification problem shows an F1 score. A regression problem has a mean squared error (MSE) score.

When you create a quick model chart, you select a dataset you want evaluated, and a target label against which you want feature importance to be compared. Data Wrangler does the following:

- Infers the data types for the target label and each feature in the dataset selected.
- Determines the problem type. Based on the number of distinct values in the label column, Data Wrangler determines if this is a regression or classification problem type. Data Wrangler sets a

categorical threshold to 100. If there are more than 100 distinct values in the label column, Data Wrangler classifies it as a regression problem; otherwise, it is classified as a classification problem.

- Pre-process features and label data for training. The algorithm used requires encoding features to vector type and encoding labels to double type.
- Trains a random forest algorithm with 70% of data. Spark's [RandomForestRegressor](#) is used to train a model for regression problems. The [RandomForestClassifier](#) is used to train a model for classification problems.
- Evaluates a random forest model with the remaining 30% of data. Data Wrangler evaluates classification models using an F1 score and evaluates regression models using an MSE score.
- Calculates feature importance for each feature using the Gini importance method.

Below is the user interface for the Quick Model feature.



## Target Leakage

Target leakage occurs when there is data in a machine learning training dataset that is strongly correlated with the target label, but is not available in real-world data. For example, you may have a column in your dataset that serves as a proxy for the column you want to predict with your model.

When you use the **Target Leakage** analysis, you specify the following:

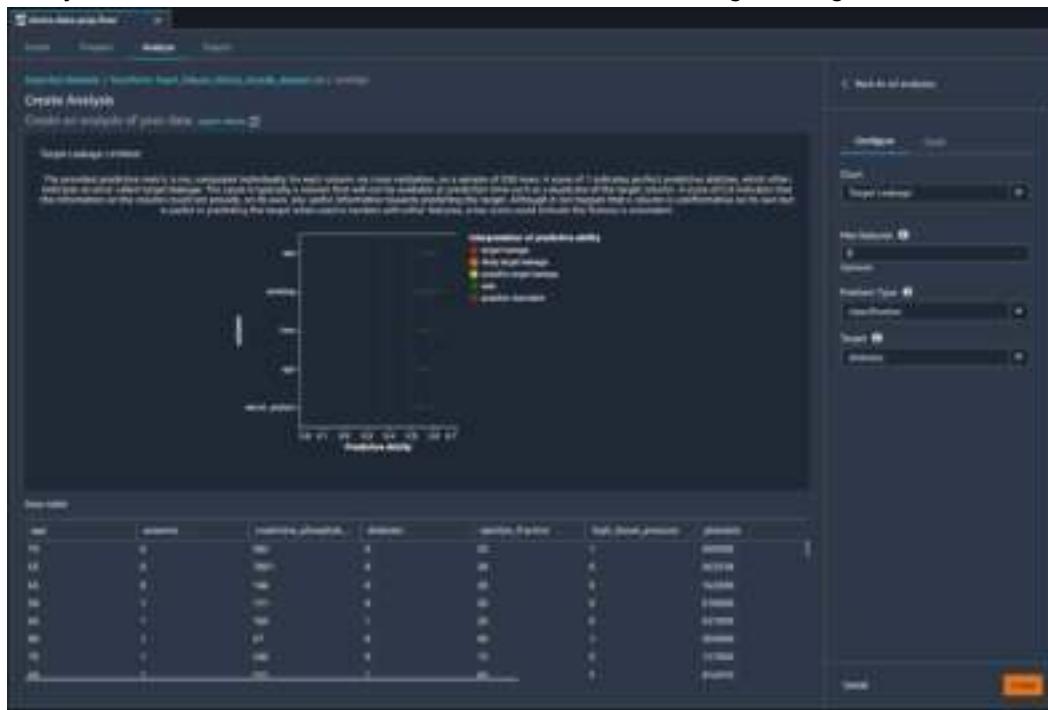
- **Target:** This is the feature about which you want your ML model to be able to make predictions.
- **Problem type:** This is the ML problem type on which you are working. Problem type can either be **classification** or **regression**.
- (Optional) **Max features:** This is the maximum number of features to present in the visualization, which shows features ranked by their risk of being target leakage.

For classification, the Target Leakage analysis uses the area under the receiver operating characteristic, or AUC - ROC curve for each column, up to **Max features**. For regression, it uses a coefficient of determination, or R2 metric.

The AUC - ROC curve provides a predictive metric, computed individually for each column using cross-validation, on a sample of up to around 1000 rows. A score of 1 indicates perfect predictive abilities, which often indicates target leakage. A score of 0.5 or lower indicates that the information on the

column could not provide, on its own, any useful information towards predicting the target. Although it can happen that a column is uninformative on its own but is useful in predicting the target when used in tandem with other features, a low score could indicate the feature is redundant.

For example, the following image shows a target leakage report for a diabetes classification problem, that is, predicting if a person has diabetes or not. An AUC - ROC curve is used to calculate the predictive ability of five features, and all are determined to be safe from target leakage.



# Bias Report

You can use the bias report in Data Wrangler to uncover potential biases in your data. To generate a bias report, you must specify the target column, or **Label**, that you want to predict and a **Facet**, or the column that you want to inspect for biases.

**Label** – The feature about which you want a model to make predictions. For example, if you are predicting customer conversion, you may select a column containing data on whether or not a customer has placed an order. You must also specify whether this feature is a label or a threshold. If you specify a label, you must specify what a *positive outcome* looks like in your data. In the customer conversion example, a positive outcome may be a 1 in the orders column, representing the positive outcome of a customer placing an order within the last three months. If you specify a threshold, you must specify a lower bound defining a positive outcome. For example, if your customer orders columns contains the number of orders placed in the last year, you may want to specify 1.

**Facet** – The column that you want to inspect for biases. For example, if you are trying to predict customer conversion, your facet may be the age of the customer. You may choose this facet because you believe that your data is biased toward a certain age group. You must identify whether the facet is measured as a value or threshold. For example, if you wanted to inspect one or more specific ages, you select **Value** and specify those ages. If you want to look at an age group, you select **Threshold** and specify the threshold of ages you want to inspect.

After you select your feature and label, you select the types of bias metrics you want to calculate.

To learn more, see [Generate reports for bias in pre-training data](#).

## Create Custom Visualizations

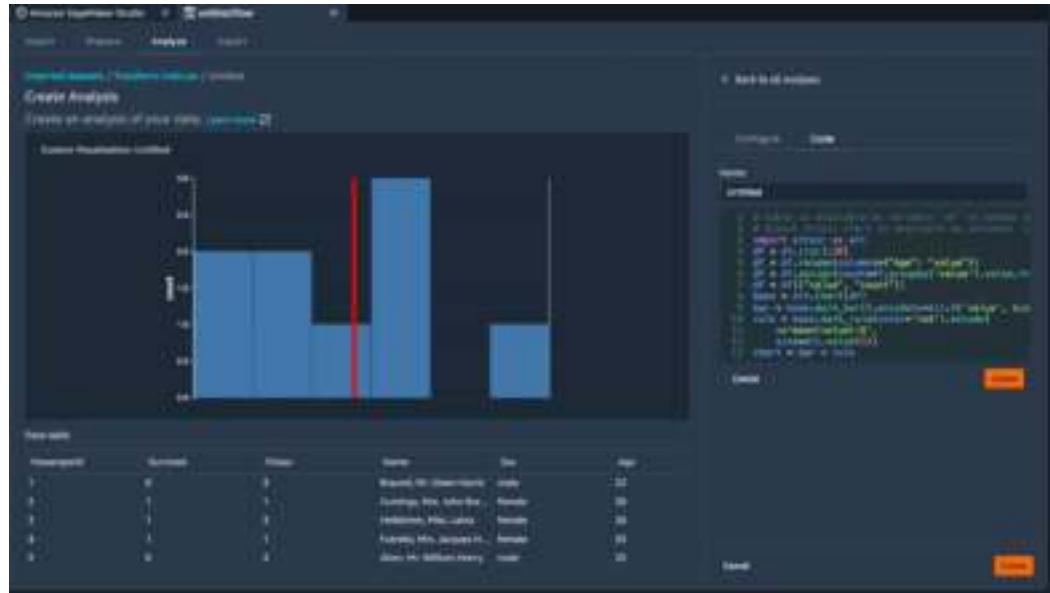
Use the **Code** tab to create custom visualizations. Your dataset, after undergoing the most recently added transformation, is available as a [Pandas DataFrame](#) in this code block via the `df` variable.

You must provide an output variable, `chart`, to store an [Altair](#) output chart. For example, the following is an example of a code block that can be used to generate a custom histogram using the titanic dataset.

```
import altair as alt
df = df.iloc[:30]
df = df.rename(columns={"Age": "value"})
df = df.assign(count=df.groupby('value').value.transform('count'))
df = df[['value', 'count']]
base = alt.Chart(df)
bar = base.mark_bar().encode(x=alt.X('value', bin=True, axis=None), y=alt.Y('count'))
rule = base.mark_rule(color='red').encode(
    x='mean(value):Q',
    size=alt.value(5))
chart = bar + rule
```

### To create a custom visualization:

1. In the **Visualize** area of Data Wrangler, choose the **Code** tab.
2. Give your visualization a **Name**.
3. Enter your code in the code box.
4. Choose **Preview** to preview your visualization.
5. Choose **Add Visualization** to add your visualization.



## Export

When you create a Data Wrangler data flow, you can choose from four export options to easily integrate that data flow into your data processing pipeline. Data Wrangler offers export options to SageMaker **Data Wrangler Job**, **Pipeline**, **Python code**, **Feature Store** and **Amazon S3**.

The following options create a Jupyter Notebook to run your data flow and integrate with the respective SageMaker feature.

- **Data Wrangler Job**
- **Pipeline**
- **Feature Store**
- **Amazon S3**

For these options, you choose one or more steps in your data flow to export. When you select a step, all downstream steps are also exported. For example, if you have defined seven consecutive steps in your data flow, and you choose to export the 7th step, code is exported to run steps 1 through 7. The Jupyter Notebooks automatically open when you select one of these export options, and you can run the notebook directly in Studio using a **Python 3 (Data science)** kernel.

If you select **Python code**, a Python file is created containing all steps in your data flow.

When you choose an export option that creates a Jupyter Notebook and you run the notebook, it exports your data flow (.flow file) to the default SageMaker S3 bucket for the Amazon Region in which the data flow was created, under the prefix *data\_wrangler\_flows*. Default S3 buckets have the following naming convention: *sagemaker-region-account number*. For example, if your account number is 111122223333 and you are using Studio in us-east-1, your imported datasets are stored in *sagemaker-us-east-1-111122223333*. In this example, your .flow files created in us-east-1 are stored in *s3://sagemaker-region-account number/data\_wrangler\_flows/*.

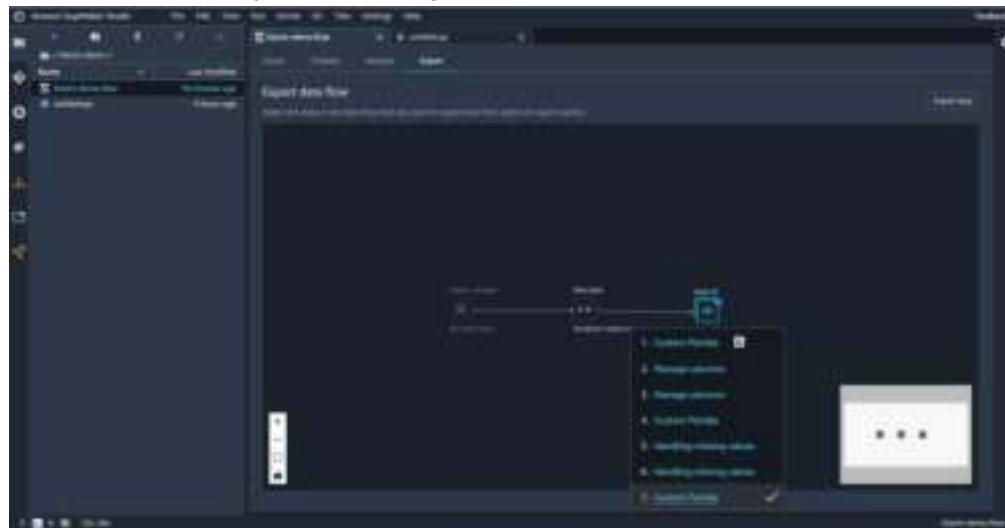
**Important**

If you do not use the IAM managed policy, `AmazonSageMakerFullAccess`, to grant Amazon roles permission to use Data Wrangler, make sure you grant these roles permission to access this bucket. See [Security and Permissions \(p. 653\)](#) for an example IAM policy you can use to grant these permissions.

Use the following procedure to export a data flow. Use the sections on this page to learn more about each export option.

**To export your Data Wrangler flow:**

1. Save your Data Wrangler flow. Select **File** and then select **Save Data Wrangler Flow**.
2. Navigate to the **Export** tab.
3. Select the last step in your Data Wrangler flow.



4. Choose **Export step**.

5. Select the export option you want.

## Export to a Data Wrangler Job

If you export a Data Wrangler job Jupyter Notebook, we recommend that you select **Python 3 (Data Science)** for the **Kernel** and run it directly in Studio to run your data flow and process your data. Follow the instructions in the notebook to launch your Data Wrangler job.

Data Wrangler jobs use processing jobs to process your data. You can run these processing jobs using `ml.m5.4x1`, `ml.m5.12x1`, and `ml.m5.24x1` instances and support one instance count. By default, the notebook that is exported from Data Wrangler sets the following `instance_count` and `instance_type`:

```
instance_count = 1
instance_type = "ml.m5.xlarge"
```

You can monitor your Data Wrangler job status in the [SageMaker console](#) in the **Processing** tab. The processing job is named `data-wrangler-flow-processing-flow_id`. The `flow_id` is defined in the notebook using the day and time the notebook is ran.

Additionally, you can monitor your Data Wrangler job using Amazon CloudWatch. For additional information, see [Monitor Amazon SageMaker Processing Jobs with CloudWatch Logs and Metrics](#).

## Export to SageMaker Pipelines

When you want to build and deploy large-scale machine learning workflows, you can use SageMaker ML Pipelines to create end-to-end workflows that manage and deploy SageMaker jobs. ML Pipelines allows you to build workflows that manage your SageMaker data preparation, model training, and model deployment jobs. Because of this, you can take advantage of the first-party algorithms that SageMaker offers. To learn more about SageMaker Pipelines, see [SageMaker Pipelines](#).

When you export one or more steps from your data flow to SageMaker Pipelines, Data Wrangler creates a Jupyter Notebook that you can use to define, instantiate, run, and manage a pipeline.

### Use A Jupyter Notebook to Create a Pipeline

The Jupyter Notebook that Data Wrangler produces can be used to define a pipeline. The pipeline includes a data processing step that is defined by your data flow.

You can add additional steps to your pipeline by adding steps to the `steps` list in the following code in the notebook:

```
pipeline = Pipeline(
    name=Pipeline_name,
    parameters=[instance_type, instance_count],
    steps=[step_process], #Add more steps to this list to run in your Pipeline
)
```

To learn more about defining pipelines, see [Define SageMaker Pipeline](#).

## Export to Python Code

To export all steps in your data flow to a Python file that you can manually integrate into any data processing workflow, choose the **Export to Code** option.

You may need to configure the Python script to make it runnable. For example, you may need to modify your Spark environment, and ensure you are running the script from an environment that has permission to access Amazon resources.

## Export to the SageMaker Feature Store

The SageMaker Feature Store can be used to create, share, and manage curated data for machine learning (ML) development. You can configure an online and offline feature store to be a centralized store for features and associated metadata so features can be easily discovered and reused. To learn more about the Data Wrangler feature store, see [Amazon SageMaker Feature Store](#).

### Use A Jupyter Notebook to Add Features to a Feature Store

The Jupyter Notebook that SageMaker produces can be used to process your dataset using a SageMaker Data Wrangler job, and then ingest the data into an online and offline feature store.

#### Important

The IAM role you use to run this notebook must have the following Amazon managed policies attached: `AmazonSageMakerFullAccess` and `AmazonSageMakerFeatureStoreAccess`.

You only need to enable one online or offline feature store when you create a feature group. Optionally, you can enable both. To disable online store creation, set `EnableOnlineStore` to `False`:

```
# Online Store Configuration
online_store_config = {
    "EnableOnlineStore": False
}
```

The notebook uses the column names and types of the dataframe you export to create a feature group schema, which is used to create a feature group. A feature group is a group of features defined in the feature store to describe a record. The feature group defines the schema and features contained in the feature group. A feature group definition is composed of a list of features, a record identifier feature name, an event time feature name, and configurations for its online store and offline store.

Each feature in a feature group can have one of the following types: *String*, *Fractional*, or *Integral*. If a column in your exported dataframe is not one of these types, it defaults to *String*.

The following is an example of a feature group schema.

```
column_schema = [
    {
        "name": "Height",
        "type": "long"
    },
    {
        "name": "Input",
        "type": "string"
    },
    {
        "name": "Output",
        "type": "string"
    },
    {
        "name": "Sum",
        "type": "string"
    },
    {
        "name": "Time",
        "type": "string"
    }
]
```

Additionally, you must specify a record identifier name and event time feature name:

- The record identifier name is the name of the feature whose value uniquely identifies a record defined in the feature store. Only the latest record per identifier value is stored in the online store. The record identifier feature name must be one of feature definitions' names.
- The event time feature name is the name of the feature that stores the `EventTime` of a record in a feature group. An `EventTime` is a point in time when a new event occurs that corresponds to the creation or update of a record in a feature. All records in the feature group must have a corresponding `EventTime`.

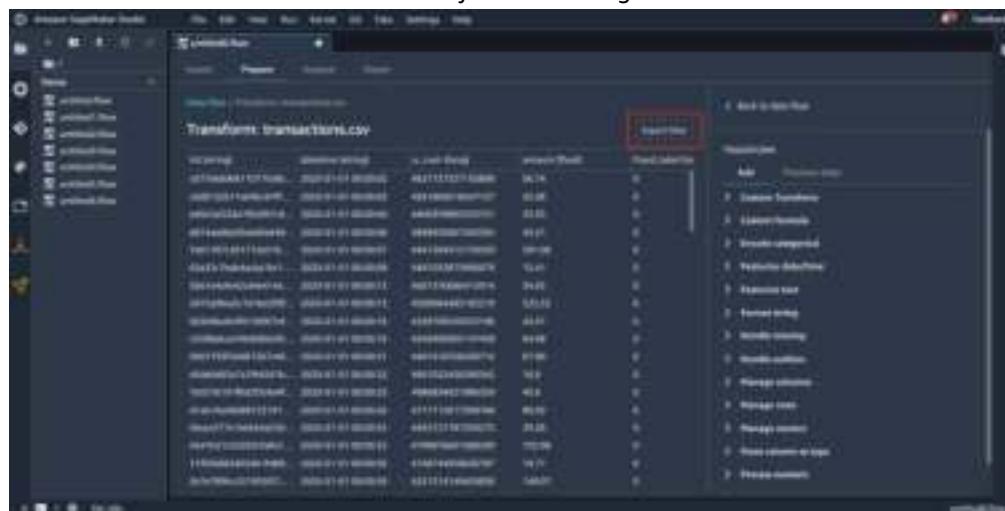
The notebook uses these configurations to create a feature group, process your data at scale, and then ingest the processed data into your online and offline feature stores. To learn more, see [Data Sources and Ingestion](#).

## Export to Amazon S3

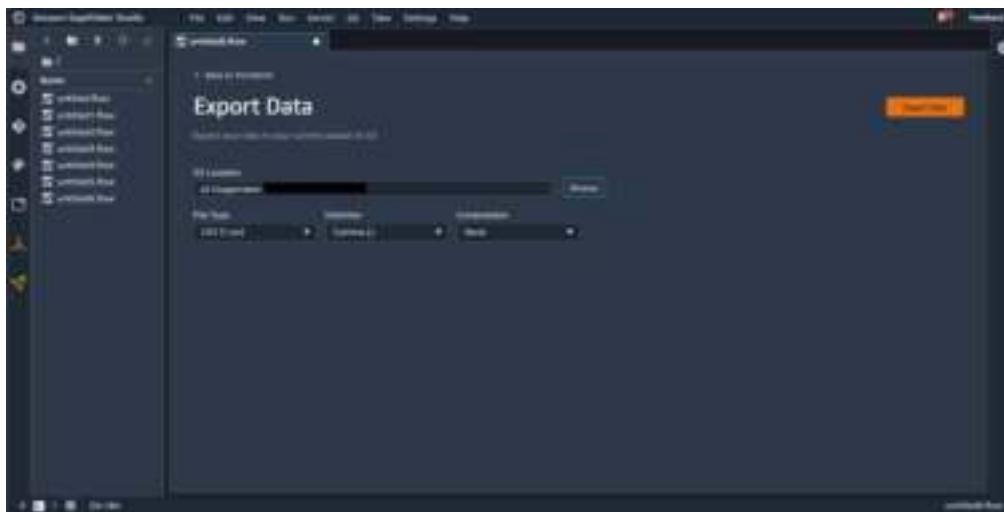
Data Wrangler allows you to export your data directly to an Amazon S3 bucket.

### Save your data to Amazon S3

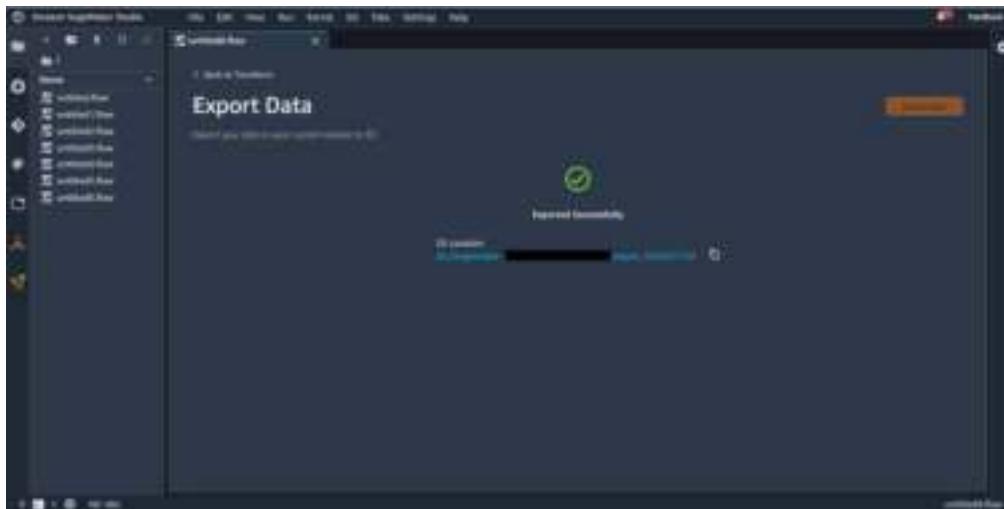
1. After you have imported your data into Data Wrangler, choose **Export Data** under the **Prepare** tab to save the current transformed dataset you are viewing.



2. An **Export Data** screen will appear as shown in the following screenshot. You can save your data to the default Amazon S3 bucket. You can also use the search bar to modify and select an Amazon S3 bucket path of your choosing. You can choose different options for **File Type**, **Delimiter**, and **Compression**. After you complete setting up your bucket path and options, select **Export Data**.



3. A progress bar will appear to show the status of data export. After it is successfully exported and saved to the Amazon S3 bucket, success screen appears as shown in the following screenshot.

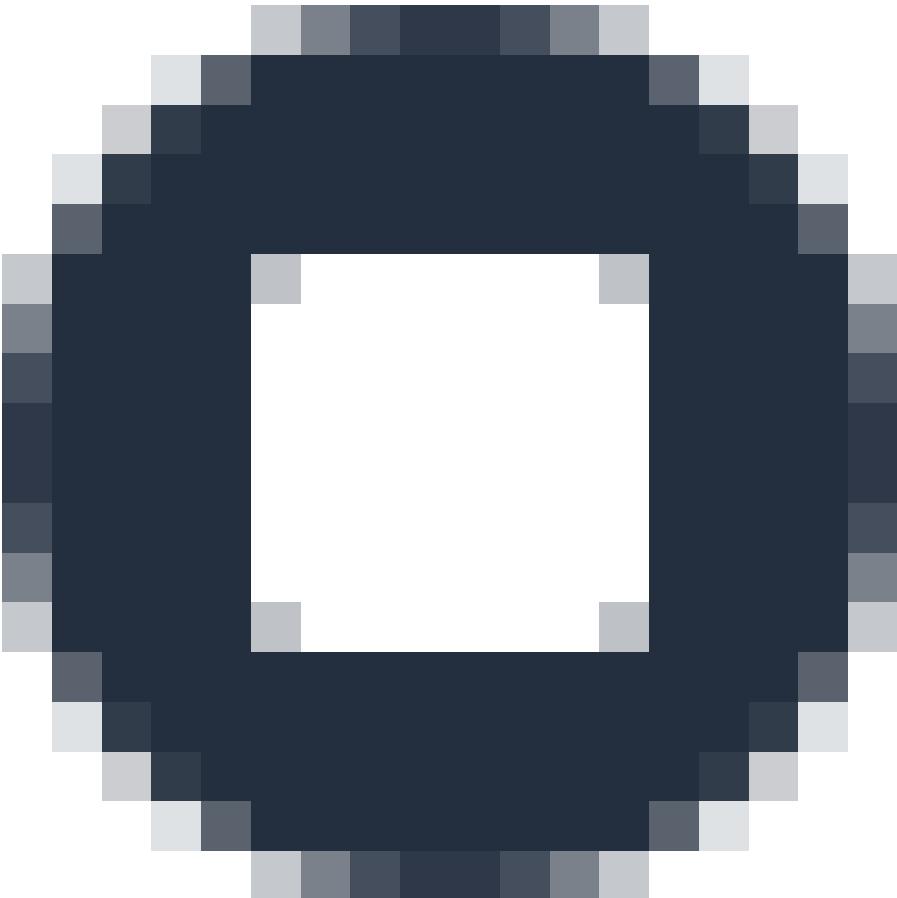


## Shut Down Data Wrangler

When you are not using Data Wrangler, it is important to shut down the instance on which it runs to avoid incurring additional fees.

To avoid losing work, save your data flow before shutting Data Wrangler down. To save your data flow in Studio, select **File** and then select **Save Data Wrangler Flow**. Note that Data Wrangler automatically saves your data flow every 60 seconds.

### To shut down the Data Wrangler instance in Studio:

1. In Studio, select the **Running Instances and Kernels** icon (  ).

2. Under **RUNNING APPS** is the **sagemaker-data-wrangler-1.0** app. Select the shut down icon next to this app (  ).

Data Wrangler runs on an ml.m5.4xlarge instance. This instance disappears from **RUNNING INSTANCES** when you shut down the Data Wrangler app.

After you shut down the Data Wrangler app, it has to restart the next time you open a Data Wrangler flow file. This can take a few minutes.

## Update Data Wrangler

To update Data Wrangler to the latest release, you must first shut down the corresponding KernelGateway app from the SageMaker Studio Control Panel. After the KernelGateway app is shut down, you must restart it by opening a new or existing Data Wrangler flow in SageMaker Studio. When you open a new or existing Data Wrangler flow, the kernel that starts contains the latest version of Data Wrangler.

### Update your Studio and Data Wrangler instance

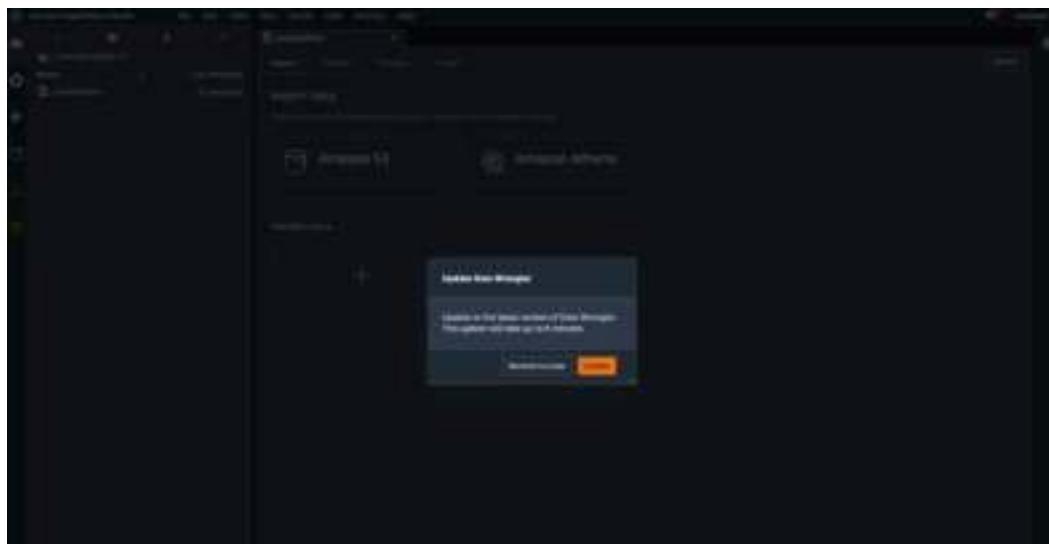
1. Navigate to your [SageMaker Console](#).
2. Choose SageMaker and then SageMaker Studio.

3. Choose your user name.
4. Under **Apps**, in the row displaying the **App name**, choose **Delete app** for the app that starts with sagemaker-data-wrang, and the JupyterServer app.
5. Choose **Yes, delete app**.
6. Type delete in the confirmation box.
7. Choose **Delete**.
8. Reopen your Studio instance. When you begin to create a Data Wrangler flow, your instance will now use the latest version of Data Wrangler.

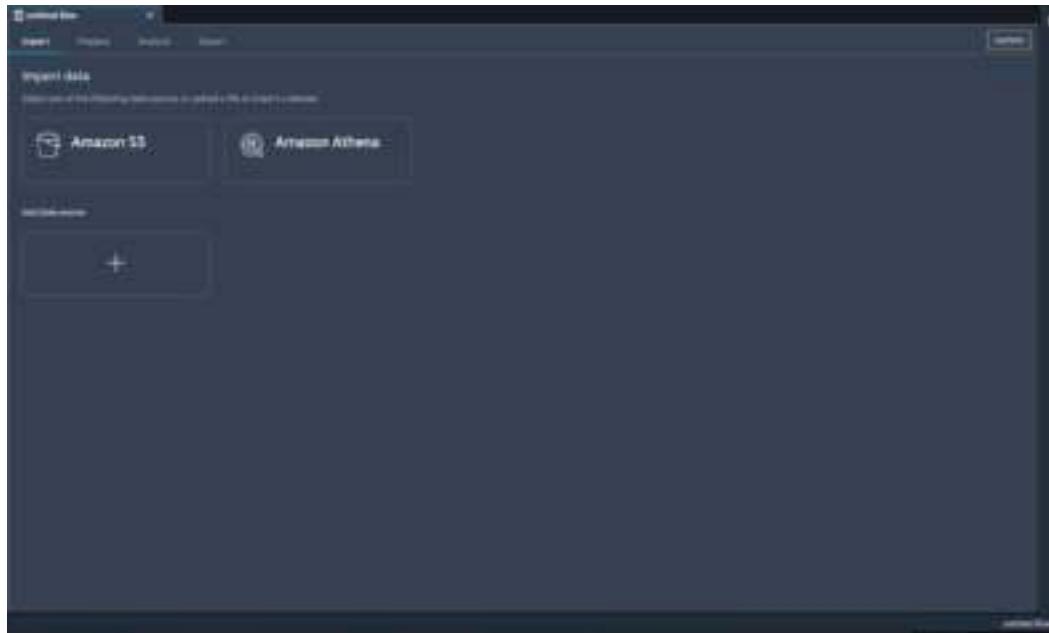
Alternatively, if you are using a Data Wrangler application version that is not the latest version, and you have an existing Data Wrangler flow open, you are prompted to update your Data Wrangler application version in the Studio UI. Below is a screenshot showing this prompt.

**Important**

Note that this will update the Data Wrangler Kernel gateway app only. You still need to shut down the JupyterServer app in your user account. To do this, follow the steps above.



You can also select Remind me later in which case an Update button will appear in the top right corner of the screen.



## Security and Permissions

When you query data from Athena or Amazon Redshift, the queried dataset is automatically stored in the default SageMaker S3 bucket for the Amazon Region in which you are using Studio. Additionally, when you export a Jupyter Notebook from Amazon SageMaker Data Wrangler and run it, your data flows, or .flow files, are saved to the same default bucket, under the prefix `data_wrangler_flows`.

For high-level security needs, you can configure a bucket policy that restricts the Amazon roles that have access to this default SageMaker S3 bucket. Use the following section to add this type of policy to an S3 bucket. To follow the instructions on this page, use the Amazon Command Line Interface (Amazon CLI). To learn how, see [Configuring the Amazon CLI](#) in the IAM User Guide.

Additionally, you need to grant each IAM role that uses Data Wrangler permissions to access required resources. If you do not require granular permissions for the IAM role you use to access Data Wrangler, you can add the IAM managed policy, [AmazonSageMakerFullAccess](#), to an IAM role that you use to create your Studio user. This policy grants you full permission to use Data Wrangler. If you require more granular permissions, refer to the section, [Grant an IAM Role Permission to Use Data Wrangler \(p. 654\)](#).

## Add a Bucket Policy To Restrict Access to Datasets Imported to Data Wrangler

You can add a policy to the S3 bucket that contains your Data Wrangler resources using an Amazon S3 bucket policy. Resources that Data Wrangler uploads to your default SageMaker S3 bucket in the Amazon Region you are using Studio in include the following:

- Queried Amazon Redshift results. These are stored under the `redshift/` prefix.
- Queried Athena results. These are stored under the `athena/` prefix.
- The .flow files uploaded to Amazon S3 when you run an exported Jupyter Notebook Data Wrangler produces. These are stored under the `data_wrangler_flows/` prefix.

Use the following procedure to create an S3 bucket policy that you can add to restrict IAM role access to that bucket. To learn how to add a policy to an S3 bucket, see [How do I add an S3 Bucket policy?](#).

### To set up a bucket policy on the S3 bucket that stores your Data Wrangler resources:

1. Configure one or more IAM roles that you want to be able to access Data Wrangler.
2. Open a command prompt or shell. For each role that you create, replace `role-name` with the name of the role and run the following:

```
$ aws iam get-role --role-name role-name
```

In the response, you'll see a `RoleId` string which begins will AROA. Copy this string.

3. Add the following policy to the SageMaker default bucket in the Amazon Region in which you are using Data Wrangler. Replace `region` with the Amazon Region in which the bucket is located, and `account-id` with your Amazon account ID. Replace `userIds` starting with `AROAEXAMPLEID` with the IDs of an Amazon roles to which you want to grant permission to use Data Wrangler.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::sagemaker-region-account-id/data_wrangler_flows/",
        "arn:aws:s3:::sagemaker-region-account-id/data_wrangler_flows/*",
        "arn:aws:s3:::sagemaker-region-account-id/athena",
        "arn:aws:s3:::sagemaker-region-account-id/athena/*",
        "arn:aws:s3:::sagemaker-region-account-id/redshift",
        "arn:aws:s3:::sagemaker-region-account-id/redshift/*"
      ],
      "Condition": {
        "StringNotLike": {
          "aws:userId": [
            "AROAEXAMPLEID_1:*",
            "AROAEXAMPLEID_2:*"
          ]
        }
      }
    ]
  }
}
```

## Grant an IAM Role Permission to Use Data Wrangler

You can grant an IAM role permission to use Data Wrangler with the general IAM managed policy, [AmazonSageMakerFullAccess](#). This is a general policy that includes permissions required to use all SageMaker services. This policy grants an IAM role full access to Data Wrangler. You should be aware of the following when using [AmazonSageMakerFullAccess](#) to grant access to Data Wrangler:

- If you import data from Amazon Redshift, the **Database User** name must have the prefix `sagemaker_access`.
- This managed policy only grants permission to access buckets with one of the following in the name: `SageMaker`, `SageMaker`, `sagemaker`, or `aws-glue`. If want to use Data Wrangler to import from an S3 bucket without these phrases in the name, refer to the last section on this page to learn how to grant permission to an IAM entity to access your S3 buckets.

If you have high-security needs, you can attach the policies in this section to an IAM entity to grant permissions required to use Data Wrangler.

If you have datasets in Amazon Redshift or Athena that an IAM role needs to import from Data Wrangler, you must add a policy to that entity to access these resources. The following policies are the most restrictive policies you can use to give an IAM role permission to import data from Amazon Redshift and Athena.

To learn how to attach a custom policy to an IAM role, refer to [Managing IAM policies](#) in the IAM User Guide.

#### **Policy example to grant access to an Athena dataset import**

The following policy assumes that the IAM role has permission to access the underlying S3 bucket where data is stored through a separate IAM policy.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "athena>ListDataCatalogs",
                "athena>ListDatabases",
                "athena>ListTableMetadata",
                "athena>GetQueryExecution",
                "athena>GetQueryResults",
                "athena>StartQueryExecution",
                "athena>StopQueryExecution"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue>CreateTable"
            ],
            "Resource": [
                "arn:aws:glue::::table/*/sagemaker_tmp_*",
                "arn:aws:glue::::table/sagemaker_featurestore/*",
                "arn:aws:glue::::catalog",
                "arn:aws:glue::::database/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue>DeleteTable"
            ],
            "Resource": [
                "arn:aws:glue::::table/*/sagemaker_tmp_*",
                "arn:aws:glue::::catalog",
                "arn:aws:glue::::database/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "glue>GetDatabases",
                "glue>GetTable",
                "glue>GetTables"
            ],
        }
    ]
}
```

```

        "Resource": [
            "arn:aws:glue::::table/*",
            "arn:aws:glue::::catalog",
            "arn:aws:glue::::database/*"
        ]
    },
    {
        "Effect": "Allow",
        "Action": [
            "glue>CreateDatabase",
            "glue:GetDatabase"
        ],
        "Resource": [
            "arn:aws:glue::::catalog",
            "arn:aws:glue::::database/sagemaker_featurestore",
            "arn:aws:glue::::database/sagemaker_processing",
            "arn:aws:glue::::database/default",
            "arn:aws:glue::::database/sagemaker_data_wrangler"
        ]
    }
]
}

```

#### **Policy example to grant access to an Amazon Redshift dataset import**

The following policy grants permission to set up an Amazon Redshift connection to Data Wrangler using database users that have the prefix `sagemaker_access` in the name. To grant permission to connect using additional database users, add additional entries under "Resources" in the following policy. The following policy assumes that the IAM role has permission to access the underlying S3 bucket where data is stored through a separate IAM policy, if applicable.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "redshift-data:ExecuteStatement",
                "redshift-data:DescribeStatement",
                "redshift-data:CancelStatement",
                "redshift-data:GetStatementResult",
                "redshift-data>ListSchemas",
                "redshift-data>ListTables"
            ],
            "Resource": [
                "*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "redshift:GetClusterCredentials"
            ],
            "Resource": [
                "arn:aws:redshift::::dbuser:/sagemaker_access*/",
                "arn:aws:redshift::::dbname:/*"
            ]
        }
    ]
}

```

#### **Policy to grant access to an S3 bucket**

If your dataset is stored in Amazon S3, you can grant an IAM role permission to access this bucket with a policy similar to the following. This example grants programmatic read-write access to the bucket named **test**:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": ["s3>ListBucket"],
            "Resource": ["arn:aws:s3:::test"]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3>PutObject",
                "s3>GetObject",
                "s3>DeleteObject"
            ],
            "Resource": ["arn:aws:s3:::test/*"]
        }
    ]
}
```

To import data from Athena and Amazon Redshift, you must grant an IAM role permission to access the following prefixes under the default Amazon S3 bucket in the Amazon Region Data Wrangler is being used in: `athena/`, `redshift/`. If a default Amazon S3 bucket does not already exist in the Amazon Region, you must also give the IAM role permission to create a bucket in this region.

Additionally, if you want the IAM role to be able to use the SageMaker Feature Store, SageMaker Pipeline, and Data Wrangler job export options, you must grant access to the prefix `data_wrangler_flows/` in this bucket.

Data Wrangler uses the `athena/` and `redshift/` prefixes to store preview files and imported datasets. To learn more, see [Imported Data Storage \(p. 617\)](#).

Data Wrangler uses the `data_wrangler_flows/` prefix to store .flow files when you run a Jupyter Notebook exported from Data Wrangler. To learn more, see [Export \(p. 645\)](#).

Use a policy similar to the following to grant the permissions described above:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3>GetObject",
                "s3>PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::sagemaker-region-account-id/data_wrangler_flows/",
                "arn:aws:s3:::sagemaker-region-account-id/data_wrangler_flows/*",
                "arn:aws:s3:::sagemaker-region-account-id/athena",
                "arn:aws:s3:::sagemaker-region-account-id/athena/*",
                "arn:aws:s3:::sagemaker-region-account-id/redshift",
                "arn:aws:s3:::sagemaker-region-account-id/redshift/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [

```

```

        "s3:CreateBucket",
        "s3>ListBucket"
    ],
    "Resource": "arn:aws:s3:::sagemaker-region-account-id"
},
{
    "Effect": "Allow",
    "Action": [
        "s3>ListAllMyBuckets",
        "s3:GetBucketLocation"
    ],
    "Resource": "*"
}
]
}

```

You can also access data in your Amazon S3 bucket from another Amazon account by specifying the Amazon S3 bucket URL. To do this, the IAM policy that grants access to the Amazon S3 bucket in the other account should use a policy similar to this below where `BucketFolder` is the specific directory in the users bucket `UserBucket`. This policy should be added to the user granting access to their bucket for another user.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3GetObject",
                "s3PutObject",
                "s3PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::UserBucket/BucketFolder/*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3ListBucket"
            ],
            "Resource": "arn:aws:s3:::UserBucket",
            "Condition": {
                "StringLike": {
                    "s3Prefix": [
                        "BucketFolder/*"
                    ]
                }
            }
        }
    ]
}

```

The user that will be accessing bucket (not the bucket owner) will need to add a policy similar to this below to their user. Note that `AccountX` and `TestUser` below refers to the bucket owner and their User respectively.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::AccountX:user/TestUser"
            },

```

```

        "Action": [
            "s3:GetObject",
            "s3:PutObject",
            "s3:PutObjectAcl"
        ],
        "Resource": [
            "arn:aws:s3:::UserBucket/BucketFolder/*"
        ]
    },
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws:iam::AccountX:user/TestUser"
        },
        "Action": [
            "s3>ListBucket"
        ],
        "Resource": [
            "arn:aws:s3:::UserBucket"
        ]
    }
]
}

```

#### Policy example to grant access to use SageMaker Studio

Use a policy like to the following to create an IAM execution role that can be used to set up a Studio instance.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker>CreatePresignedDomainUrl",
                "sagemaker>DescribeDomain",
                "sagemaker>ListDomains",
                "sagemaker>DescribeUserProfile",
                "sagemaker>ListUserProfiles",
                "sagemaker:*App",
                "sagemaker>ListApps"
            ],
            "Resource": "*"
        }
    ]
}

```

## Snowflake and Data Wrangler

All permissions for Amazon resources will be managed via your IAM role attached to your SageMaker Studio instance. Snowflake specific permissions are to be managed by the Snowflake admin, as they can grant granular permissions/privileges to each Snowflake user. This includes databases, schemas, tables, warehouses, and storage integration objects. You will have to ensure that the correct permissions are set up outside of Data Wrangler.

Note that the Snowflake `COPY INTO Amazon S3` command moves data from Snowflake to S3 over the public internet by default but data in transit will be secured using SSL. Data at rest in Amazon S3 will be encrypted with SSE-KMS using the default Amazon-managed CMK.

With respect to Snowflake credentials storage, Data Wrangler does not store customer credentials. Data Wrangler uses Amazon Secrets Manager to store the credentials in a Secret and rotates secrets

as part of a best practice security plan. The Snowflake or Studio administrator needs to ensure that the data scientist's Studio execution role is granted permission to perform `GetSecretValue` on the Secret where the credentials are stored. If already attached to the Studio execution role, the `AmazonSageMakerFullAccess` policy will have the necessary permissions to read secrets created by Data Wrangler and secrets created by following the naming and tagging convention in the instructions above. Secrets that do not follow the conventions will need to be separately granted access. It is recommended to use Secrets Manager to prevent sharing credentials over unsecured channels, however note that a logged-in user can retrieve the plain-text password by launching a terminal or Python notebook in Studio and then invoking API calls from the Secrets Manager API.

## Data Encryption with KMS-CMK

For files stored in Amazon S3 that have server-side encryption enabled and the encryption type is SSE-KMS, the SageMaker Studio user role will need to be added as a Key user for them to be able to decrypt the file and import in Data Wrangler.

Below is a screenshot that shows a Studio user role added as a Key user. See [IAM Roles](#) to access Users under the left panel to make this change.

The screenshot shows the 'Key users' section of the AWS IAM console. At the top, there is a search bar. Below it, a table lists three entries:

	Name
<input type="checkbox"/>	AmazonSageMaker-ExecutionRole-20210409T160134
<input type="checkbox"/>	Admin

## Amazon S3 CMK setup for Data Wrangler imported data storage

By default, Data Wrangler uses Amazon S3 buckets that have the following naming convention: `sagemaker-region-account number`. For example, if your account number is `111122223333` and you are using Studio in `us-east-1`, your imported datasets are stored with the following naming convention: `sagemaker-us-east-1-111122223333`.

Below are instructions on how to setup a custom managed CMK for your default Amazon S3 bucket.

1. To enable server-side encryption and setup custom managed CMK for your default S3 bucket, see [Using KMS Encryption](#).
2. After following Step 1, navigate to KMS in your Amazon Web Services Management Console. Find the CMK you selected in Step 1 of the previous step, and add the Studio role as the key user. To do this, follow the [Allows key users to use the CMK](#).

## Release Notes

Data Wrangler is regularly updated with new features and bug fixes. To upgrade the version of Data Wrangler you are using in Studio, follow the instructions in [Update Studio Apps \(p. 118\)](#).

### Release Notes

#### 4/26/2021

##### Enhancements:

- Added support for distributed Processing Jobs. You can use multiple instances when running a processing job.
- Data Wrangler Processing job now automatically coalesces small outputs when estimated result size is less than 1 gigabytes.
- Feature Store Notebook: Improved feature store ingestion performance
- Data Wrangler Processing job now use 1.x as the authoritative container tag for future releases.

##### Bug Fixes:

- Fixed rendering issues for faceted histogram.
- Fixed Export to Processing Job to support vector type columns.
- Fixed "Extract using regex" operator to return the first captured group if one or more exists in the regular expression or regex.

#### 2/8/2021

##### New Functionalities:

- Data Wrangler Flows supports multiple instances.
- Updated Export to Data Wrangler Job Notebook to use SageMaker SDK 2.20.0.
- Updated Export to Pipeline Notebook to use SageMaker SDK 2.20.0.
- Updated Export to Pipeline Notebook to add XGBoost training example as an optional step.

##### Enhancements:

- To improve performance, importing CSV files that contain multiple lines in a single field is no longer supported.

##### Bug Fixes:

- Fixed type inference issue in Quick model.
- Fixed the bias metric bug in bias reports.
- Fixed the Featurize text transform to work with columns with missing values.
- Fixed Histogram and Scatter plot built-in visualizations to work with datasets that contain array-like columns.
- Athena query now re-runs if the query execution ID has expired.

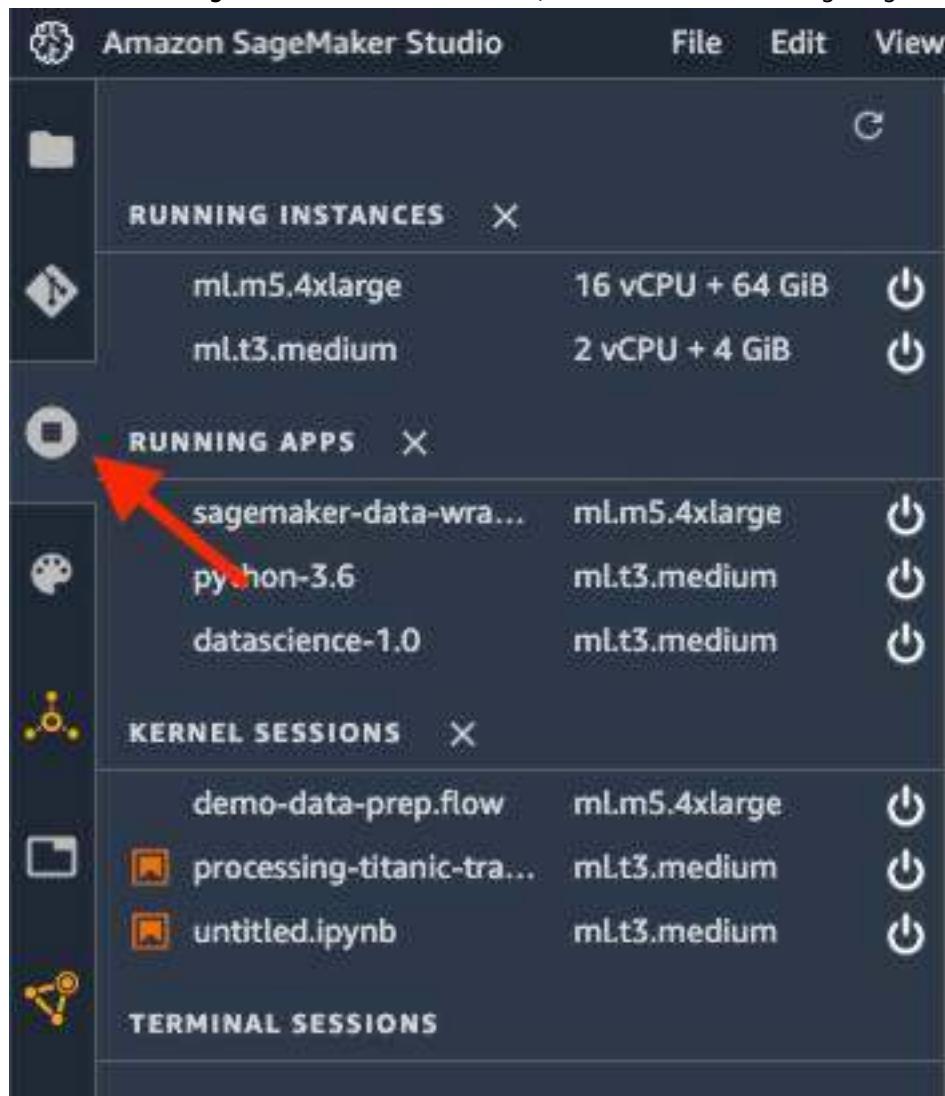
## Troubleshoot

If an issue arises when using Amazon SageMaker Data Wrangler, we recommend you do the following:

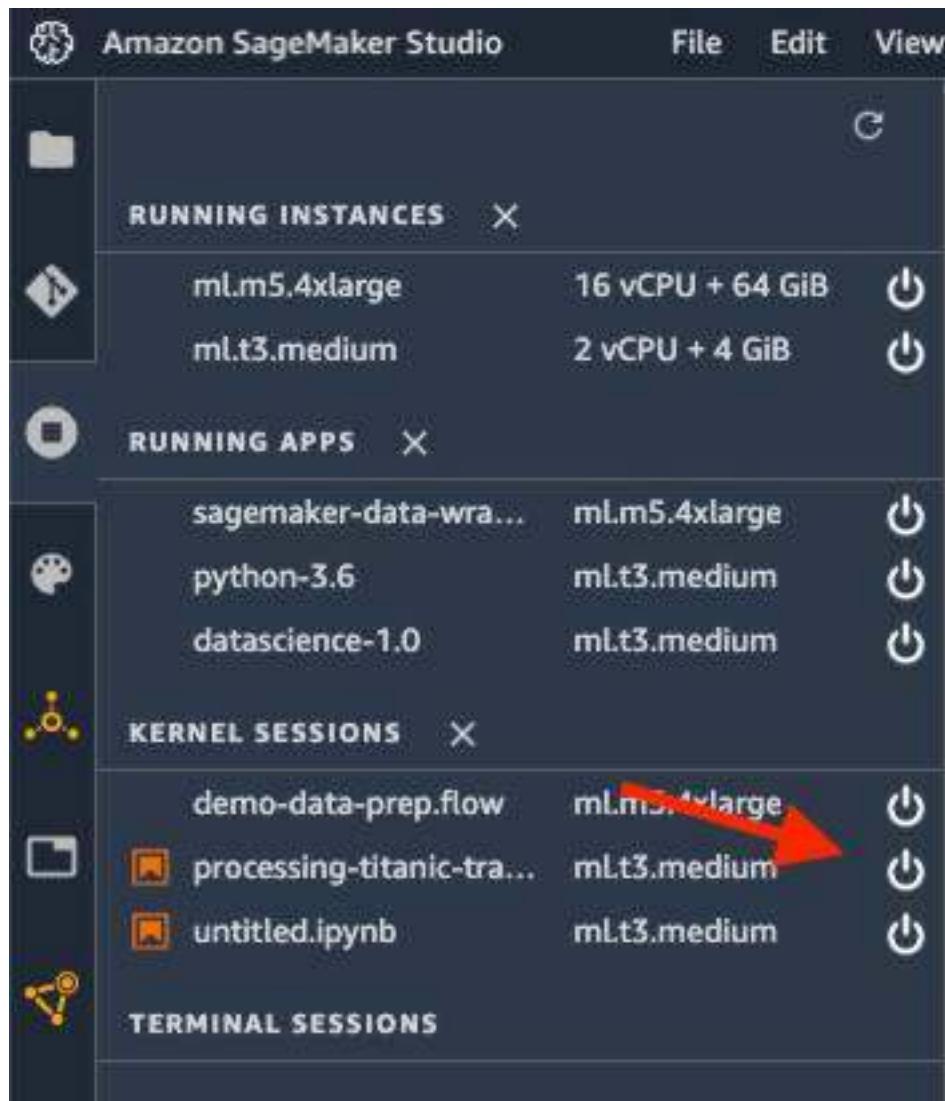
- If an error message is provided, read the message and resolve the issue it reports if possible.
- Make sure the IAM role of your Studio user has the required permissions to perform the action. Reference [Security and Permissions \(p. 653\)](#).
- If the issue occurs when you are trying to import from another Amazon service, such as Amazon Redshift or Athena, make sure that you have configured the necessary permissions and resources to perform the data import. Reference [Import \(p. 588\)](#).

As a last resort, you can try restarting the kernel on which Data Wrangler is running.

1. Save and exit out of the .flow file for which you want to restart the kernel.
2. Select the **Running Terminals and Kernels** icon, as shown in the following image.



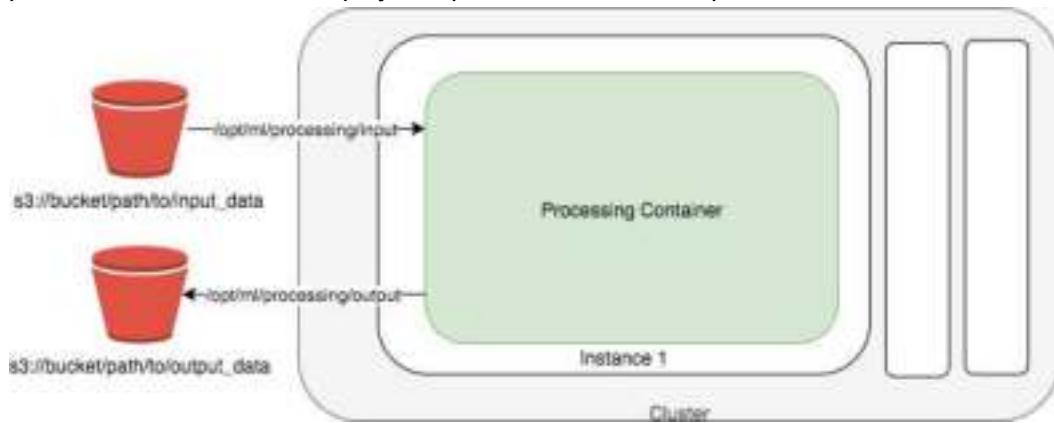
3. Select the **Stop** icon to the right of the .flow file for which you want to terminate the kernel, as shown in the following image.



4. Refresh the browser.
5. Reopen the .flow file on which you were working.

# Process Data

To analyze data and evaluate machine learning models on Amazon SageMaker, use Amazon SageMaker Processing. With Processing, you can use a simplified, managed experience on SageMaker to run your data processing workloads, such as feature engineering, data validation, model evaluation, and model interpretation. You can also use the Amazon SageMaker Processing APIs during the experimentation phase and after the code is deployed in production to evaluate performance.



The preceding diagram shows how Amazon SageMaker spins up a Processing job. Amazon SageMaker takes your script, copies your data from Amazon Simple Storage Service (Amazon S3), and then pulls a processing container. The processing container image can either be an Amazon SageMaker built-in image or a custom image that you provide. The underlying infrastructure for a Processing job is fully managed by Amazon SageMaker. Cluster resources are provisioned for the duration of your job, and cleaned up when a job completes. The output of the Processing job is stored in the Amazon S3 bucket you specified.

**Note**

Your data must be stored in an Amazon S3 bucket.

## Use Amazon SageMaker Processing Sample Notebooks

We provide two sample Jupyter notebooks that show how to perform data preprocessing, model evaluation, or both.

For a sample notebook that shows how to run scikit-learn scripts to perform data preprocessing and model training and evaluation with the SageMaker Python SDK for Processing, see [scikit-learn Processing](#). This notebook also shows how to use your own custom container to run processing workloads with your Python libraries and other specific dependencies.

For a sample notebook that shows how to use Amazon SageMaker Processing to perform distributed data preprocessing with Spark, see [Distributed Processing \(Spark\)](#). This notebook also shows how to train a regression model using XGBoost on the preprocessed dataset.

For instructions on how to create and access Jupyter notebook instances that you can use to run these samples in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). After you have

created a notebook instance and opened it, choose the **SageMaker Examples** tab to see a list of all the SageMaker samples. To open a notebook, choose its **Use** tab and choose **Create copy**.

## Monitor Amazon SageMaker Processing Jobs with CloudWatch Logs and Metrics

Amazon SageMaker Processing provides Amazon CloudWatch logs and metrics to monitor processing jobs. CloudWatch provides CPU, GPU, memory, GPU memory, and disk metrics, and event logging. For more information, see [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 2523\)](#) and [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 2534\)](#).

## Data Processing with Apache Spark

Apache Spark is a unified analytics engine for large-scale data processing. Amazon SageMaker provides prebuilt Docker images that include Apache Spark and other dependencies needed to run distributed data processing jobs. With the [Amazon SageMaker Python SDK](#), you can easily apply data transformations and extract features (feature engineering) using the Spark framework. For information about using the SageMaker Python SDK to run Spark processing jobs, see [Data Processing with Spark](#) in the [Amazon SageMaker Python SDK](#).

A code repository that contains the source code and Dockerfiles for the Spark images is available on [GitHub](#).

## Running a Spark Processing Job

You can use the `sagemaker.spark.PySparkProcessor` or `sagemaker.spark.SparkJarProcessor` class to run your Spark application inside of a processing job. Note you can set `MaxRuntimeInSeconds` to a maximum runtime limit of 5 days. With respect to execution time, and number of instances used, simple spark workloads see a near linear relationship between the number of instances vs. time to completion.

The following code example shows how to run a processing job that invokes your PySpark script `preprocess.py`.

```
from sagemaker.spark.processing import PySparkProcessor

spark_processor = PySparkProcessor(
    base_job_name="spark-preprocessor",
    framework_version="2.4",
    role=role,
    instance_count=2,
    instance_type="ml.m5.xlarge",
    max_runtime_in_seconds=1200,
)

spark_processor.run(
    submit_app="preprocess.py",
    arguments=['s3_input_bucket', bucket,
               's3_input_key_prefix', input_prefix,
               's3_output_bucket', bucket,
               's3_output_key_prefix', output_prefix]
)
```

For an in-depth look, see the [Distributed Data Processing with Apache Spark and SageMaker Processing example notebook](#).

If you are not using the [Amazon SageMaker Python SDK](#) and one of its Processor classes to retrieve the pre-built images, you can retrieve these images yourself. The SageMaker prebuilt Docker images are stored in Amazon Elastic Container Registry (Amazon ECR). For a complete list of the available pre-built Docker images, see the [available images](#) document.

To learn more about using the SageMaker Python SDK with Processing containers, see [Amazon SageMaker Python SDK](#).

## Data Processing with scikit-learn

For a sample notebook that shows how to run scikit-learn scripts using a Docker image provided and maintained by SageMaker to preprocess data and evaluate models, see [scikit-learn Processing](#). To use this notebook, you need to install the SageMaker Python SDK for Processing.

This notebook runs a processing job using `SKLearnProcessor` class from the the SageMaker Python SDK to run a scikit-learn script that you provide. The script preprocesses data, trains a model using a SageMaker training job, and then runs a processing job to evaluate the trained model. The processing job estimates how the model is expected to perform in production.

To learn more about using the SageMaker Python SDK with Processing containers, see the [SageMaker Python SDK](#).

The following code example shows how the notebook uses `SKLearnProcessor` to run your own scikit-learn script using a Docker image provided and maintained by SageMaker, instead of your own Docker image.

```
from sagemaker.sklearn.processing import SKLearnProcessor
from sagemaker.processing import ProcessingInput, ProcessingOutput

sklearn_processor = SKLearnProcessor(framework_version='0.20.0',
                                      role=role,
                                      instance_type='ml.m5.xlarge',
                                      instance_count=1)

sklearn_processor.run(code='preprocessing.py',
                      inputs=[ProcessingInput(
                          source='s3://path/to/my/input-data.csv',
                          destination='/opt/ml/processing/input')],
                      outputs=[ProcessingOutput(source='/opt/ml/processing/output/train'),
                               ProcessingOutput(source='/opt/ml/processing/output/
validation'),
                               ProcessingOutput(source='/opt/ml/processing/output/test')]
)
```

To process data in parallel using Scikit-Learn on Amazon SageMaker Processing, you can shard input objects by S3 key by setting `s3_data_distribution_type='ShardedByS3Key'` inside a `ProcessingInput` so that each instance receives about the same number of input objects.

## Use Your Own Processing Code

You can install libraries to run your scripts in your own processing container or, in a more advanced scenario, you can build your own processing container that satisfies the contract to run in Amazon

SageMaker. For a formal specification that defines the contract for an Amazon SageMaker Processing container, see [Build Your Own Processing Container \(Advanced Scenario\) \(p. 668\)](#).

### Topics

- [Run Scripts with Your Own Processing Container \(p. 667\)](#)
- [Build Your Own Processing Container \(Advanced Scenario\) \(p. 668\)](#)

## Run Scripts with Your Own Processing Container

You can use scikit-learn scripts to preprocess data and evaluate your models. To see how to run scikit-learn scripts to perform these tasks see the [scikit-learn Processing](#) sample notebook. This notebook uses the ScriptProcessor class from the Amazon SageMaker Python SDK for Processing.

The following example shows how to use a `ScriptProcessor` class to run a Python script with your own image that runs a processing job that processes input data, and saves the processed data in Amazon Simple Storage Service (Amazon S3).

The notebook shows the general workflow for using a `ScriptProcessor` class.

1. Create a Docker directory and add the Dockerfile used to create the processing container. Install pandas and scikit-learn into it. (You could also install your own dependencies with a similar `RUN` command.)

```
mkdir docker

%%writefile docker/Dockerfile

FROM python:3.7-slim-buster

RUN pip3 install pandas==0.25.3 scikit-learn==0.21.3
ENV PYTHONUNBUFFERED=TRUE

ENTRYPOINT ["python3"]
```

2. Build the container using the `docker` command, create an Amazon Elastic Container Registry (Amazon ECR) repository, and push the image to Amazon ECR.

```
import boto3

account_id = boto3.client('sts').get_caller_identity().get('Account')
region = boto3.Session().region_name
ecr_repository = 'sagemaker-processing-container'
tag = ':latest'
processing_repository_uri = '{}.dkr.ecr.{}.amazonaws.com/{}'.format(account_id, region,
    ecr_repository + tag)

# Create ECR repository and push docker image
!docker build -t $ecr_repository docker
!aws ecr get-login-password --region $region | docker login --username Amazon --
password-stdin $account_id.dkr.ecr.$region.amazonaws.com
!aws ecr create-repository --repository-name $ecr_repository
!docker tag $ecr_repository $processing_repository_uri
!docker push $processing_repository_uri
```

3. Set up the `ScriptProcessor` from the SageMaker Python SDK to run the script.

```
from sagemaker.processing import ScriptProcessor, ProcessingInput, ProcessingOutput
script_processor = ScriptProcessor(command=['python3'],
```

```
image_uri='<image_uri>',
role='<role_arn>',
instance_count=1,
instance_type='ml.m5.xlarge')
```

4. Run the script.

```
script_processor.run(code='preprocessing.py',
                     inputs=[ProcessingInput(
                         source='s3://path/to/my/input-data.csv',
                         destination='/opt/ml/processing/input')],
                     outputs=[ProcessingOutput(source='/opt/ml/processing/output/
train'),
                           ProcessingOutput(source='/opt/ml/processing/output/
validation'),
                           ProcessingOutput(source='/opt/ml/processing/output/test')])
```

You can use the same procedure with any other library or system dependencies. You can also use existing Docker images. This includes images that you run on other platforms such as [Kubernetes](#).

## Build Your Own Processing Container (Advanced Scenario)

You can provide Amazon SageMaker Processing with a Docker image that has your own code and dependencies to run your data processing, feature engineering, and model evaluation workloads.

The following example of a Dockerfile builds a container with the Python libraries scikit-learn and pandas, which you can run as a processing job.

```
FROM python:3.7-slim-buster

# Install scikit-learn and pandas
RUN pip3 install pandas==0.25.3 scikit-learn==0.21.3

# Add a Python script and configure Docker to run it
ADD processing_script.py /
ENTRYPOINT ["python3", "/processing_script.py"]
```

Build and push this Docker image to an Amazon Elastic Container Registry (Amazon ECR) repository and ensure that your SageMaker IAM role can pull the image from Amazon ECR. Then you can run this image on Amazon SageMaker Processing.

## How Amazon SageMaker Processing Runs Your Processing Container Image

Amazon SageMaker Processing runs your processing container image in a similar way as the following command, where `AppSpecification.ImageUri` is the Amazon ECR image URI that you specify in a `CreateProcessingJob` operation.

```
docker run [AppSpecification.ImageUri]
```

This command runs the `ENTRYPOINT` command configured in your Docker image.

You can also override the entrypoint command in the image or give command-line arguments to your entrypoint command using the `AppSpecification.ContainerEntrypoint` and

`AppSpecification.ContainerArgument` parameters in your `CreateProcessingJob` request. Specifying these parameters configures Amazon SageMaker Processing to run the container similar to the way that the following command does.

```
docker run --entry-point [AppSpecification.ContainerEntrypoint]  
[AppSpecification.ImageUri] [AppSpecification.ContainerArguments]
```

For example, if you specify the `ContainerEntrypoint` to be `[python3, -v, /processing_script.py]` in your `CreateProcessingJob` request, and `ContainerArguments` to be `[data-format, csv]`, Amazon SageMaker Processing runs your container with the following command.

```
python3 -v /processing_script.py data-format csv
```

When building your processing container, consider the following details:

- Amazon SageMaker Processing decides whether the job completes or fails depending on the exit code of the command run. A processing job completes if all of the processing containers exit successfully with an exit code of 0, and fails if any of the containers exits with a non-zero exit code.
- Amazon SageMaker Processing lets you override the processing container's entrypoint and set command-line arguments just like you can with the Docker API. Docker images can also configure the entrypoint and command-line arguments using the `ENTRYPOINT` and `CMD` instructions. The way `CreateProcessingJob`'s `ContainerEntrypoint` and `ContainerArgument` parameters configure a Docker image's entrypoint and arguments mirrors how Docker overrides the entrypoint and arguments through the Docker API:
  - If neither `ContainerEntrypoint` nor `ContainerArguments` are provided, Processing uses the default `ENTRYPOINT` or `CMD` in the image.
  - If `ContainerEntrypoint` is provided, but not `ContainerArguments`, Processing runs the image with the given entrypoint, and ignores the `ENTRYPOINT` and `CMD` in the image.
  - If `ContainerArguments` is provided, but not `ContainerEntrypoint`, Processing runs the image with the default `ENTRYPOINT` in the image and with the provided arguments.
  - If both `ContainerEntrypoint` and `ContainerArguments` are provided, Processing runs the image with the given entrypoint and arguments, and ignores the `ENTRYPOINT` and `CMD` in the image.
- You must use the exec form of the `ENTRYPOINT` instruction in your Dockerfile (`ENTRYPOINT ["executable", "param1", "param2"]`) instead of the shell form (`ENTRYPOINT command param1 param2`). This lets your processing container receive `SIGINT` and `SIGKILL` signals, which Processing uses to stop processing jobs with the `StopProcessingJob` API.
- `/opt/ml` and all its subdirectories are reserved by SageMaker. When building your Processing Docker image, don't place any data required by your processing container in these directories.
- If you plan to use GPU devices, make sure that your containers are nvidia-docker compatible. Include only the CUDA toolkit in containers. Don't bundle NVIDIA drivers with the image. For more information about nvidia-docker, see [NVIDIA/nvidia-docker](#).

## How Amazon SageMaker Processing Configures Input and Output For Your Processing Container

When you create a processing job using the `CreateProcessingJob` operation, you can specify multiple `ProcessingInput` and `ProcessingOutput` values.

You use the `ProcessingInput` parameter to specify an Amazon Simple Storage Service (Amazon S3) URI to download data from, and a path in your processing container to download the data to. The `ProcessingOutput` parameter configures a path in your processing container from which to

upload data, and where in Amazon S3 to upload that data to. For both `ProcessingInput` and `ProcessingOutput`, the path in the processing container must begin with `/opt/ml/processing/`.

For example, you might create a processing job with one `ProcessingInput` parameter that downloads data from `s3://your-data-bucket/path/to/input/csv/data` into `/opt/ml/processing/csv` in your processing container, and a `ProcessingOutput` parameter that uploads data from `/opt/ml/processing/processed_csv` to `s3://your-data-bucket/path/to/output/csv/data`. Your processing job would read the input data, and write output data to `/opt/ml/processing/processed_csv`. Then it uploads the data written to this path to the specified Amazon S3 output location.

**Important**

Symbolic links (symlinks) can not be used to upload output data to Amazon S3. Symlinks are not followed when uploading output data.

## How Amazon SageMaker Processing Provides Logs and Metrics for Your Processing Container

When your processing container writes to `stdout` or `stderr`, Amazon SageMaker Processing saves the output from each processing container and puts it in Amazon CloudWatch logs. For information about logging, see [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 2534\)](#).

Amazon SageMaker Processing also provides CloudWatch metrics for each instance running your processing container. For information about metrics, see [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 2523\)](#).

## How Amazon SageMaker Processing Configures Your Processing Container

Amazon SageMaker Processing provides configuration information to your processing container through environment variables and two JSON files—`/opt/ml/config/processingjobconfig.json` and `/opt/ml/config/resourceconfig.json`—at predefined locations in the container.

When a processing job starts, it uses the environment variables that you specified with the `Environment` map in the `CreateProcessingJob` request. The `/opt/ml/config/processingjobconfig.json` file contains information about the hostnames of your processing containers, and is also specified in the `CreateProcessingJob` request.

The following example shows the format of the `/opt/ml/config/processingjobconfig.json` file.

```
{  
    "ProcessingJobArn": "<processing_job_arn>",  
    "ProcessingJobName": "<processing_job_name>",  
    "AppSpecification": {  
        "ImageUri": "<image_uri>",  
        "ContainerEntrypoint": null,  
        "ContainerArguments": null  
    },  
    "Environment": {  
        "KEY": "VALUE"  
    },  
    "ProcessingInputs": [  
        {  
            "InputName": "input-1",  
            "S3Input": {  
                "LocalPath": "/opt/ml/processing/input/dataset",  
                "S3Uri": "<s3_uri>",  
                "S3DataDistributionType": "FullyReplicated",  
                "S3ObjectKeyFilter": "  
                    <filter>  
                "
```

```

        "S3DataType": "S3Prefix",
        "S3InputMode": "File",
        "S3CompressionType": "None",
        "S3DownloadMode": "StartOfJob"
    }
}
],
"ProcessingOutputConfig": {
    "Outputs": [
        {
            "OutputName": "output-1",
            "S3Output": {
                "LocalPath": "/opt/ml/processing/output/dataset",
                "S3Uri": "<s3_uri>",
                "S3UploadMode": "EndOfJob"
            }
        }
    ],
    "KmsKeyId": null
},
"ProcessingResources": {
    "ClusterConfig": {
        "InstanceCount": 1,
        "InstanceType": "ml.m5.xlarge",
        "VolumeSizeInGB": 30,
        "VolumeKmsKeyId": null
    }
},
"RoleArn": "<IAM role>",
"StoppingCondition": {
    "MaxRuntimeInSeconds": 86400
}
}
}

```

The `/opt/ml/config/resourceconfig.json` file contains information about the hostnames of your processing containers. Use the following hostnames when creating or running distributed processing code.

```
{
    "current_host": "algo-1",
    "hosts": ["algo-1", "algo-2", "algo-3"]
}
```

Don't use the information about hostnames contained in `/etc/hostname` or `/etc/hosts` because it might be inaccurate.

Hostname information might not be immediately available to the processing container. We recommend adding a retry policy on hostname resolution operations as nodes become available in the cluster.

## Save and Access Metadata Information About Your Processing Job

To save metadata from the processing container after exiting it, containers can write UTF-8 encoded text to the `/opt/ml/output/message` file. After the processing job enters any terminal status ("Completed", "Stopped", or "Failed"), the "ExitMessage" field in [DescribeProcessingJob](#) contains the first 1 KB of this file. Access that initial part of file with a call to [DescribeProcessingJob](#), which returns it through the `ExitMessage` parameter. For failed processing jobs, you can use this field to communicate information about why the processing container failed.

### Important

Don't write sensitive data to the `/opt/ml/output/message` file.

If the data in this file isn't UTF-8 encoded, the job fails and returns a `ClientError`. If multiple containers exit with an `ExitMessage`, the content of the `ExitMessage` from each processing container is concatenated, then truncated to 1 KB.

## Run Your Processing Container Using the SageMaker Python SDK

You can use the SageMaker Python SDK to run your own processing image by using the `Processor` class. The following example shows how to run your own processing container with one input from Amazon Simple Storage Service (Amazon S3) and one output to Amazon S3.

```
from sagemaker.processing import Processor, ProcessingInput, ProcessingOutput

processor = Processor(image_uri='<your_ecr_image_uri>',
                      role=role,
                      instance_count=1,
                      instance_type="ml.m5.xlarge")

processor.run(inputs=[ProcessingInput(
    source='<s3_uri or local path>',
    destination='/opt/ml/processing/input_data')],
              outputs=[ProcessingOutput(
    source='/opt/ml/processing/processed_data',
    destination='<s3_uri>'),
              ])
```

Instead of building your processing code into your processing image, you can provide a `ScriptProcessor` with your image and the command that you want to run, along with the code that you want to run inside that container. For an example, see [Run Scripts with Your Own Processing Container \(p. 667\)](#).

You can also use the scikit-learn image that Amazon SageMaker Processing provides through `SKLearnProcessor` to run scikit-learn scripts. For an example, see [Data Processing with scikit-learn \(p. 666\)](#).

# Create, Store, and Share Features with Amazon SageMaker Feature Store

The machine learning (ML) development process often begins with extracting data signals also known as *features* from data to train ML models. Amazon SageMaker Feature Store makes it easy for data scientists, machine learning engineers, and general practitioners to create, share, and manage features for machine learning (ML) development. Feature Store accelerates this process by reducing repetitive data processing and curation work required to convert raw data into features for training an ML algorithm.

Further, the processing logic for your data is authored only once, and features generated are used for both training and inference, reducing the training-serving skew. Feature Store is a centralized store for features and associated metadata so features can be easily discovered and reused. You can create an online or an offline store. The online store is used for low latency real-time inference use cases, and the offline store is used for training and batch inference.

The following diagram shows how you can use Amazon SageMaker Feature Store as part of your machine learning pipeline. First, you read in your raw data and process it. You can ingest data via streaming to the online and offline store, or in batches directly to the offline store. You first create a `FeatureGroup` and configure it to an online or offline store, or both. Then, you can ingest data into your `FeatureGroup` and store it in your store. A `FeatureGroup` is a group of features that is defined via a schema in Feature Store to describe a record.

Online store is primarily designed for supporting real-time predictions that need low millisecond latency reads and high throughput writes. Offline store is primarily intended for batch predictions and model training. Offline store is an append only store and can be used to store and access historical feature data. The offline store can help you store and serve features for exploration and model training. The online store retains only the latest feature data. Feature group definitions are immutable after they are created.



## How Feature Store Works

In Feature Store, features are stored in a collection called a *feature group*. You can visualize a feature group as a table in which each column is a feature, with a unique identifier for each row. In principle, a

feature group is composed of features and values specific to each feature. A `Record` is a collection of values for features that correspond to a unique `RecordIdentifier`. Altogether, a `FeatureGroup` is a group of features defined in your `FeatureStore` to describe a `Record`.

You can use Feature Store in the following modes:

- **Online** – In online mode, features are read with low latency (milliseconds) reads and used for high throughput predictions. This mode requires a feature group to be stored in an online store.
- **Offline** – In offline mode, large streams of data are fed to an offline store, which can be used for training and batch inference. This mode requires a feature group to be stored in an offline store. The offline store uses your S3 bucket for storage and can also fetch data using Athena queries.
- **Online and Offline** – This includes both online and offline modes.

You can ingest data into feature groups in Feature Store in two ways: streaming or in batches. When you ingest data through streaming, a collection of records are pushed to Feature Store by calling a synchronous `PutRecord` API call. This API enables you to maintain the latest feature values in Feature Store and to push new feature values as soon an update is detected.

Alternatively, Feature Store can process and ingest data in batches. You can author features using Amazon SageMaker Data Wrangler, create feature groups in Feature Store and ingest features in batches using a SageMaker Processing job with a notebook exported from Data Wrangler. This mode allows for batch ingestion into the offline store. It also supports ingestion into the online store if the feature group is configured for both online and offline use.

## Create Feature Groups

To ingest features into Feature Store, you must first define the feature group and the feature definitions (feature name and data type) for all features that belong to the feature group. After they are created, feature groups are immutable. Feature group names are unique within an Amazon Region and Amazon account. When creating a feature group, you can also create the metadata for the feature group, such as a short description, storage configuration, features for identifying each record, and the event time, as well as tags to store information such as the author, data source, version, and more.

## Find, Discover, and Share Features

After you create a feature group in Feature Store, other authorized users of the feature store can share and discover it. Users can browse through a list of all feature groups in Feature Store or discover existing feature groups by searching by feature group name, description, record identifier name, creation date, and tags.

## Real-Time Inference for Features Stored in the Online Store

With Feature Store, you can enrich your features stored in the online store in real time with data from a streaming source (clean stream data from another application) and serve the features with low millisecond latency for real-time inference.

You can also perform joins across different `FeatureGroups` for real-time inference by querying two different `FeatureGroups` in the client application.

# Offline Store for Model Training and Batch Inference

Feature Store provides offline storage for feature values in your S3 bucket. Your data is stored in your S3 bucket using a prefixing scheme based on event time. The offline store is an append-only store, enabling Feature Store to maintain a historical record of all feature values. Data is stored in the offline store in Parquet format for optimized storage and query access.

You can query, explore, and visualize features using Data Wrangler from Amazon SageMaker Studio. Feature Store supports combining data to produce, train, validate, and test data sets, and allows you to extract data at different points in time.

## Feature Data Ingestion

Feature generation pipelines can be created to process large batches (1 million rows of data or more) or small batches, and to write feature data to the offline or online store. Streaming sources such as Amazon Managed Streaming for Apache Kafka or Amazon Kinesis can also be used as data sources from which features are extracted and directly fed to the online store for training, inference, or feature creation.

You can push records to Feature Store by calling the synchronous `PutRecord` API call. Since this is a synchronous API call, it allows small batches of updates to be pushed in a single API call. This enables you to maintain high freshness of the feature values and publish values as soon as an update is detected. These are also called *streaming features*.

When feature data is ingested and updated, Feature Store stores historical data for all features in the offline store. For batch ingest, you can pull feature values from your S3 bucket or use Athena to query. You can also use Data Wrangler to process and engineer new features that can then be exported to a chosen S3 bucket to be accessed by Feature Store. For batch ingestion, you can configure a processing job to batch ingest your data into Feature Store, or you can pull feature values from your S3 bucket using Athena.

## Get started with Amazon SageMaker Feature Store

To get started using Amazon SageMaker Feature Store, review the basic concepts, learn how to ingest data for your feature store, and then walk through a Feature Store example. The following sections explain how to create feature groups, ingest data into the groups, and how to manage security for your feature store.

### Topics

- [Feature Store Concepts \(p. 675\)](#)
- [Create Feature Groups \(p. 676\)](#)
- [Adding required policies to your IAM role \(p. 686\)](#)
- [Use Amazon SageMaker Feature Store with Amazon SageMaker Studio \(p. 691\)](#)

## Feature Store Concepts

The following list of terms are key to understanding the capabilities of Amazon SageMaker Feature Store:

- **Feature store** – Serves as the single source of truth to store, retrieve, remove, track, share, discover, and control access to features.

- **Feature** – A measurable property or characteristic that encapsulates an observed phenomenon. In the Amazon SageMaker Feature Store API, a feature is an attribute of a record. You can define a name and type for every feature stored in Feature Store. Name uniquely identifies a feature within a feature group. Type identifies the datatype for the values of the feature. Supported datatypes are: String, Integral and Fractional.
- **Feature group** – A `FeatureGroup` is the main Feature Store resource that contains the metadata for all the data stored in Amazon SageMaker Feature Store. A feature group is a logical grouping of features, defined in the feature store, to describe records. A feature group's definition is composed of a list of feature definitions, a record identifier name, and configurations for its online and offline store.
- **Feature definition** – A `FeatureDefinition` consists of a name and one of the following data types: an Integral, String or Fractional. A `FeatureGroup` contains a list of feature definitions.
- **Record identifier name** – Each feature group is defined with a record identifier name. The record identifier name must refer to one of the names of a feature defined in the feature group's feature definitions.
- **Record** – A `Record` is a collection of values for features for a single record identifier value. A combination of record identifier name and a timestamp uniquely identify a record within a feature group.
- **Event time** – a point in time when a new event occurs that corresponds to the creation or update of a record in a feature group. All records in the feature group must have a corresponding `Eventtime`. It can be used to track changes to a record over time. The online store contains the record corresponding to the last `Eventtime` for a record identifier name, whereas the offline store contains all historic records. Event time values must be an ISO-8601 string in the format. The following formats are supported `yyyy-MM-dd'T'HH:mm:ssZ` and `yyyy-MM-dd'T'HH:mm:ss.SSSZ` where `yyyy`, `MM`, and `dd` represent the year, month, and day respectively and `HH`, `mm`, `ss`, and if applicable, `sss` represent the hour, month, second and milliseconds respectively. `T` and `Z` are constants.
- **Online Store** – the low latency, high availability cache for a feature group that enables real-time lookup of records. The online store allows quick access to the latest value for a `Record` via the `GetRecord` API. A feature group contains an `OnlineStoreConfig` controlling where the data is stored.
- **Offline store** – the `OfflineStore`, stores historical data in your S3 bucket. It is used when low (sub-second) latency reads are not needed. For example, when you want to store and serve features for exploration, model training, and batch inference. A feature group contains an `OfflineStoreConfig` controlling where the data is stored.
- **Ingestion** – The act of populating feature groups in the feature store.

## Create Feature Groups

A `FeatureGroup` is the main Feature Store resource that contains the metadata for all the data stored in Amazon SageMaker Feature Store. A feature group is a logical grouping of features, defined in the feature store, to describe records. A feature group's definition is composed of a list of feature definitions, a record identifier name, and configurations for its online and offline store. The example code in this topic uses the SageMaker Python SDK. The underlying APIs are available for developers using other languages.

Prior to using a feature store you typically load your dataset, run transformations, and set up your features for ingestion. This process has a lot of variation and is highly dependent on your data. The example code in the following topics refer to the [Introduction to Feature Store](#), [Fraud Detection with Amazon SageMaker FeatureStore](#) example notebooks respectively. We recommend that you run this notebook in Amazon SageMaker Studio because the code in this guide is conceptual and not fully functional if copied.

Feature Store supports the following data types: `String`, `Fractional` (IEEE 64-bit floating point value), and `Integral` (Int64 - 64 bit signed integral value). The default type is set to `String`. This means that, if a column in your dataset is not a `float` or `long` type, it defaults to `String` in your feature store.

You may use a schema to describe your data's columns and data types. You pass this schema into `FeatureDefinitions`, a required parameter for a `FeatureGroup`. You can use the SageMaker Python SDK, which has automatic data type detection when you use the `load_feature_definitions` function.

## Introduction to Feature Store

The example code in this topic refers to the [Introduction to Amazon SageMaker Feature Store](#) example notebook. It is recommended that you run this notebook in Amazon SageMaker Studio because the code in this guide is conceptual and not fully functional if copied.

### Step 1: Set Up

To start using Feature Store, create SageMaker, boto3 and a Feature Store sessions. Then set up the S3 bucket you want to use for your features. This is your offline store. The following code uses the SageMaker default bucket and adds a custom prefix to it.

#### Note

The role that you use must have the following managed policies attached to it:  
`AmazonS3FullAccess` and `AmazonSageMakerFeatureStoreAccess`.

```
# SageMaker Python SDK version 2.x is required
import sagemaker
import sys

import boto3
import pandas as pd
import numpy as np
import io
from sagemaker.session import Session
from sagemaker import get_execution_role

prefix = 'sagemaker-featurestore-introduction'
role = get_execution_role()

sagemaker_session = sagemaker.Session()
region = sagemaker_session.boto_region_name
s3_bucket_name = sagemaker_session.default_bucket()
```

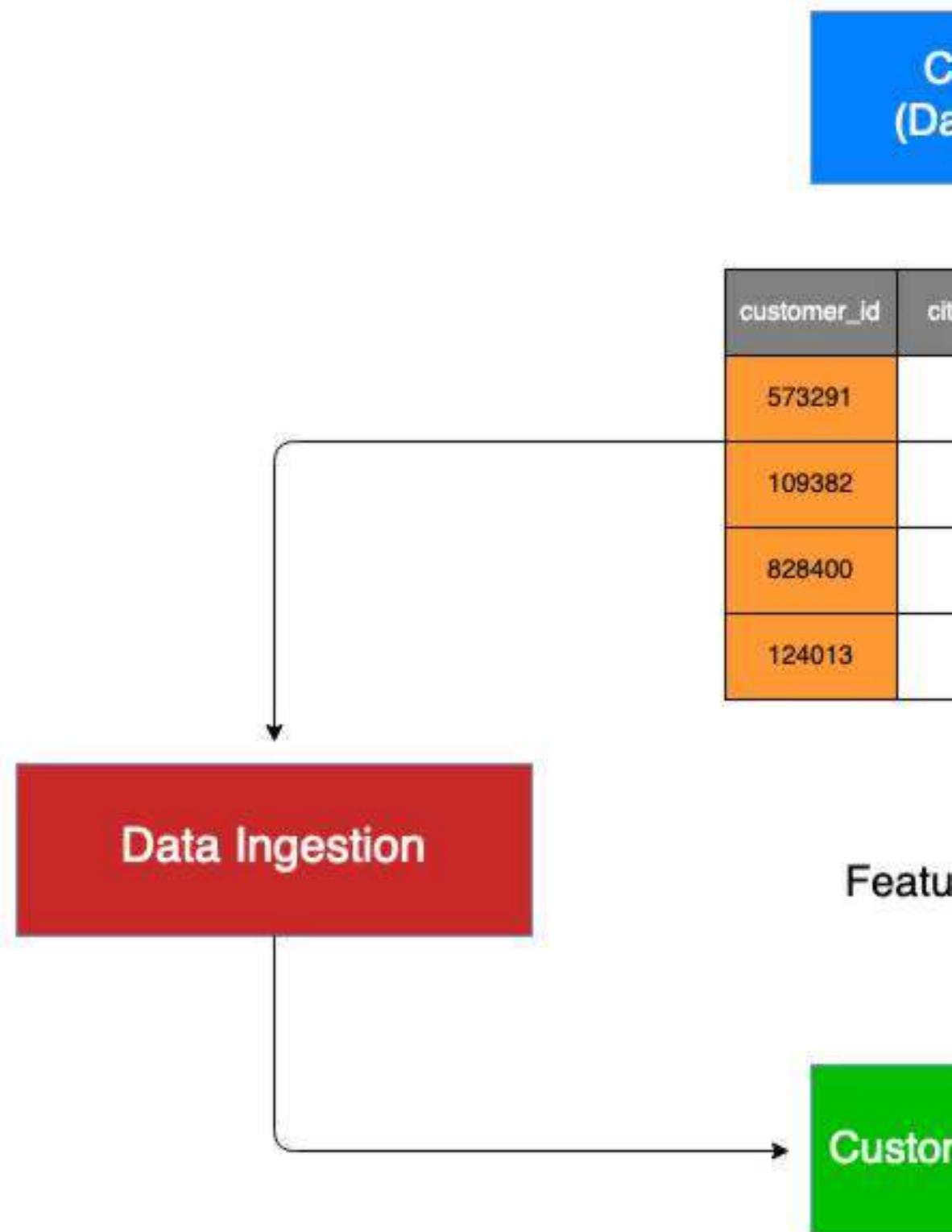
### Step 2: Inspect your data

In this notebook example we ingest synthetic data from the [Github repository](#) that hosts the full notebook.

```
customer_data = pd.read_csv("data/feature_store_introduction_customer.csv")
orders_data = pd.read_csv("data/feature_store_introduction_orders.csv")

print(customer_data.head())
print(orders_data.head())
```

The following diagram illustrates the steps the data goes through before it is ingested into Feature Store. In this notebook, we illustrate the use-case where you have data from multiple sources and want to store them independently in a feature store. Our example considers data from a data warehouse (customer data), and data from a real-time streaming service (order data).



## Step 3: Create feature groups

We first start by creating feature group names for `customer_data` and `orders_data`. Following this, we create two Feature Groups, one for `customer_data` and another for `orders_data`.

```
import time
customers_feature_group_name = 'customers-feature-group-' + strftime('%d-%H-%M-%S',
    gmtime())
orders_feature_group_name = 'orders-feature-group-' + strftime('%d-%H-%M-%S', gmtime())

current_time_sec = int(round(time.time()))
record_identifier_feature_name = "customer_id"
```

Append `EventTime` feature to your data frame. This parameter is required, and time stamps each data point.

```
customer_data["EventTime"] = pd.Series([current_time_sec]*len(customer_data),
    dtype="float64")
orders_data["EventTime"] = pd.Series([current_time_sec]*len(orders_data), dtype="float64")
```

Load feature definitions to your feature group.

```
customers_feature_group.load_feature_definitions(data_frame=customer_data)
orders_feature_group.load_feature_definitions(data_frame=orders_data)
```

Below we call `create` to create two feature groups, `customers_feature_group` and `orders_feature_group` respectively.

```
customers_feature_group.create(
    s3_uri=f"s3://{s3_bucket_name}/{prefix}",
    record_identifier_name=record_identifier_feature_name,
    event_time_feature_name="EventTime",
    role_arn=role,
    enable_online_store=True
)

orders_feature_group.create(
    s3_uri=f"s3://{s3_bucket_name}/{prefix}",
    record_identifier_name=record_identifier_feature_name,
    event_time_feature_name="EventTime",
    role_arn=role,
    enable_online_store=True
)
```

To confirm that your FeatureGroup has been created we use `DescribeFeatureGroup` and `ListFeatureGroups` APIs to display the created feature group.

```
customers_feature_group.describe()
```

```
orders_feature_group.describe()
```

```
sagemaker_session.boto_session.client('sagemaker',
    region_name=region).list_feature_groups() # We use the boto client to list FeatureGroups
```

## Step 4: Ingest data into a feature group

After the FeatureGroups have been created, we can put data into the FeatureGroups. If you are using the SageMaker Python SDK, use the `ingest` API call. If you are using by using `boto3` then use the `PutRecord` API. It will take less than 1 minute to ingest data both of these FeatureGroups. This example uses the SageMaker Python SDK, and so it uses the `ingest` API call.

```
def check_feature_group_status(feature_group):
    status = feature_group.describe().get("FeatureGroupStatus")
    while status == "Creating":
        print("Waiting for Feature Group to be Created")
        time.sleep(5)
        status = feature_group.describe().get("FeatureGroupStatus")
    print(f"FeatureGroup {feature_group.name} successfully created.")

check_feature_group_status(customers_feature_group)
check_feature_group_status(orders_feature_group)
```

```
customers_feature_group.ingest(
    data_frame=customer_data, max_workers=3, wait=True
)
```

```
orders_feature_group.ingest(
    data_frame=orders_data, max_workers=3, wait=True
)
```

Using an arbitrary customer record id, 573291 we use `get_record` to check that the data has been ingested into the feature group.

```
customer_id = 573291
sample_record = sagemaker_session.boto_session.client('sagemaker-featurestore-runtime',
    region_name=region).get_record(FeatureGroupName=customers_feature_group_name,
    RecordIdentifierAsString=str(customer_id))
```

```
sample_record
```

## Step 5: Clean up

Here we remove the Feature Groups we created.

```
customers_feature_group.delete()
orders_feature_group.delete()
```

## Step 6: Next steps

In this example notebook, you learned how to quickly get started with Feature Store, create feature groups, and ingest data into them.

For an advanced example on how to use Feature Store for a Fraud Detection use-case, see [Fraud Detection with Feature Store](#).

## Step 7: Programmers note

In this notebook we used a variety of different API calls. Most of them are accessible through the Python SDK, however some only exist within `boto3`. You can invoke the Python SDK API calls directly on your Feature Store objects, whereas to invoke API calls that exist within

boto3, you must first access a boto client through your boto and sagemaker sessions: e.g., `sagemaker_session.boto_session.client()`.

Below we list API calls used in this notebook that exist within the Python SDK and ones that exist in boto3 for your reference.

#### Python SDK API Calls

```
describe()  
ingest()  
delete()  
create()  
load_feature_definitions()
```

#### Boto3 API Calls

```
list_feature_groups()  
get_record()
```

## Fraud Detection with Feature Store

### Step 1: Set Up Feature Store

To start using Feature Store, create a SageMaker session, boto3 session, and a Feature Store session. Also, set up the S3 bucket you want to use for your features. This is your offline store. The following code uses the SageMaker default bucket and adds a custom prefix to it.

#### Note

The role that you use must have the following managed policies attached to it:  
`AmazonSageMakerFullAccess` and `AmazonSageMakerFeatureStoreAccess`.

```
import boto3  
import sagemaker  
from sagemaker.session import Session  
  
sagemaker_session = sagemaker.Session()  
region = sagemaker_session.boto_region_name  
boto_session = boto3.Session(region_name=region)  
role = sagemaker.get_execution_role()  
default_bucket = sagemaker_session.default_bucket()  
prefix = 'sagemaker-featurestore'  
offline_feature_store_bucket = 's3://{}{}'.format(default_bucket, prefix)  
  
sagemaker_client = boto_session.client(service_name='sagemaker', region_name=region)  
featurestore_runtime = boto_session.client(service_name='sagemaker-featurestore-runtime',  
region_name=region)  
  
feature_store_session = Session(  
    boto_session=boto_session,  
    sagemaker_client=sagemaker_client,  
    sagemaker_featurestore_runtime_client=featurestore_runtime  
)
```

### Step 2: Load Datasets and Partition Data into Feature Groups

Load your data into data frames for each of your features. You use these data frames after you set up the feature group. In the fraud detection example, you can see these steps in the following code.

```
import numpy as np  
import pandas as pd  
import matplotlib.pyplot as plt
```

```

import io

fraud_detection_bucket_name = 'sagemaker-featurestore-fraud-detection'
identity_file_key = 'sampled_identity.csv'
transaction_file_key = 'sampled_transactions.csv'

identity_data_object = s3_client.get_object(Bucket=fraud_detection_bucket_name,
                                             Key=identity_file_key)
transaction_data_object = s3_client.get_object(Bucket=fraud_detection_bucket_name,
                                               Key=transaction_file_key)

identity_data = pd.read_csv(io.BytesIO(identity_data_object['Body'].read()))
transaction_data = pd.read_csv(io.BytesIO(transaction_data_object['Body'].read()))

identity_data = identity_data.round(5)
transaction_data = transaction_data.round(5)

identity_data = identity_data.fillna(0)
transaction_data = transaction_data.fillna(0)

# Feature transformations for this dataset are applied before ingestion into FeatureStore.
# One hot encode card4, card6
encoded_card_bank = pd.get_dummies(transaction_data['card4'], prefix = 'card_bank')
encoded_card_type = pd.get_dummies(transaction_data['card6'], prefix = 'card_type')

transformed_transaction_data = pd.concat([transaction_data, encoded_card_type,
                                           encoded_card_bank], axis=1)
transformed_transaction_data =
    transformed_transaction_data.rename(columns={"card_bank_americane express":
                                                 "card_bank_americane express"})

```

### Step 3: Set Up Feature Groups

When you set up your feature groups, you need to customize the feature names with a unique name and set up each feature group with the `FeatureGroup` class.

```

from sagemaker.feature_store.feature_group import FeatureGroup
feature_group_name = "some string for a name"
feature_group = FeatureGroup(name=feature_group_name,
                             sagemaker_session=feature_store_session)

```

For example, in the fraud detection example, the two feature groups are `identity` and `transaction`. In the following code you can see how the names are customized with a timestamp, and then each group is set up by passing in the name and the session.

```

import time
from time import gmtime, strftime, sleep
from sagemaker.feature_store.feature_group import FeatureGroup

identity_feature_group_name = 'identity-feature-group-' + strftime('%d-%H-%M-%S', gmtime())
transaction_feature_group_name = 'transaction-feature-group-' + strftime('%d-%H-%M-%S',
                                                                       gmtime())

identity_feature_group = FeatureGroup(name=identity_feature_group_name,
                                       sagemaker_session=feature_store_session)
transaction_feature_group = FeatureGroup(name=transaction_feature_group_name,
                                         sagemaker_session=feature_store_session)

```

### Step 4: Set Up Record Identifier and Event Time Features

In this step, you specify a record identifier name and an event time feature name. This name maps to the column of the corresponding features in your data. For example, in the fraud detection example, the

column of interest is `TransactionID`. `EventTime` can be appended to your data when no timestamp is available. In the following code, you can see how these variables are set, and then `EventTime` is appended to both feature's data.

```
record_identifier_name = "TransactionID"
event_time_feature_name = "EventTime"
current_time_sec = int(round(time.time()))
identity_data[event_time_feature_name] = pd.Series([current_time_sec]*len(identity_data),
    dtype="float64")
transformed_transaction_data[event_time_feature_name] =
    pd.Series([current_time_sec]*len(transaction_data), dtype="float64")
```

## Step 5: Load Feature Definitions

You can now load the feature definitions by passing a data frame containing the feature data. In the following code for the fraud detection example, the identity feature and transaction feature are each loaded by using `load_feature_definitions`, and this function automatically detects the data type of each column of data. For developers using a schema rather than automatic detection, see the [Export Feature Groups from Data Wrangler](#) example for code that shows how to load the schema, map it, and add it as a `FeatureDefinition` that you can use to create the `FeatureGroup`. This example also covers a `boto3` implementation, which you can use instead of the SageMaker Python SDK.

```
identity_feature_group.load_feature_definitions(data_frame=identity_data); # output is suppressed
transaction_feature_group.load_feature_definitions(data_frame=transformed_transaction_data);
# output is suppressed
```

## Step 6: Create a Feature Group

In this step, you use the `create` function to create the feature group. The following code shows all of the available parameters. The online store is not created by default, so you must set this as `True` if you want to enable it. The `s3_uri` is the S3 bucket location of your offline store.

```
# create a FeatureGroup
feature_group.create(
    description = "Some info about the feature group",
    feature_group_name = feature_group_name,
    record_identifier_name = record_identifier_name,
    event_time_feature_name = event_time_feature_name,
    feature_definitions = feature_definitions,
    role_arn = role,
    s3_uri = offline_feature_store_bucket,
    enable_online_store = True,
    online_store_kms_key_id = None,
    offline_store_kms_key_id = None,
    disable_glue_table_creation = False,
    data_catalog_config = None,
    tags = ["tag1", "tag2"])
```

The following code from the fraud detection example shows a minimal `create` call for each of the two features groups being created.

```
identity_feature_group.create(
    s3_uri=offline_feature_store_bucket,
    record_identifier_name=record_identifier_name,
    event_time_feature_name=event_time_feature_name,
    role_arn=role,
    enable_online_store=True
)
```

```
transaction_feature_group.create(  
    s3_uri=offline_feature_store_bucket,  
    record_identifier_name=record_identifier_name,  
    event_time_feature_name=event_time_feature_name,  
    role_arn=role,  
    enable_online_store=True  
)
```

When you create a feature group, it takes time to load the data, and you need to wait until the feature group is created before you can use it. You can check status using the following method.

```
status = feature_group.describe().get("FeatureGroupStatus")
```

While the feature group is being created, you receive `Creating` as a response. When this step has finished successfully, the response is `Created`. Other possible statuses are `CreateFailed`, `Deleting`, or `DeleteFailed`.

## Step 7: Work with Feature Groups

Now that you've set up your feature group, you can perform any of the following tasks:

### Topics

- [Describe a Feature Group \(p. 684\)](#)
- [List Feature Groups \(p. 684\)](#)
- [Put Records in a Feature Group \(p. 684\)](#)
- [Get Records from a Feature Group \(p. 685\)](#)
- [Generate Hive DDL Commands \(p. 685\)](#)
- [Build a Training Dataset \(p. 685\)](#)
- [Write and Execute an Athena Query \(p. 686\)](#)
- [Delete a Feature Group \(p. 686\)](#)

### Describe a Feature Group

You can retrieve information about your feature group with the `describe` function.

```
feature_group.describe()
```

### List Feature Groups

You can list all of your feature groups with the `list_feature_groups` function.

```
sagemaker_client.list_feature_groups()
```

### Put Records in a Feature Group

You can use the `ingest` function to load your feature data. You pass in a data frame of feature data, set the number of workers, and choose to wait for it to return or not. The following example demonstrates using the `ingest` function.

```
feature_group.ingest(  
    data_frame=feature_data, max_workers=3, wait=True  
)
```

For each feature group you have, run the `ingest` function on the feature data you want to load.

### Get Records from a Feature Group

You can use the `get_record` function to retrieve the data for a specific feature by its record identifier. The following example uses an example identifier to retrieve the record.

```
record_identifier_value = str(2990130)
featurestore_runtime.get_record(FeatureGroupName=transaction_feature_group_name,
    RecordIdentifierValueAsString=record_identifier_value)
```

An example response from the fraud detection example:

```
...
'Record': [{ 'FeatureName': 'TransactionID', 'ValueAsString': '2990130'},
    {'FeatureName': 'isFraud', 'ValueAsString': '0'},
    {'FeatureName': 'TransactionDT', 'ValueAsString': '152647'},
    {'FeatureName': 'TransactionAmt', 'ValueAsString': '75.0'},
    {'FeatureName': 'ProductCD', 'ValueAsString': 'H'},
    {'FeatureName': 'card1', 'ValueAsString': '4577'},
...
]
```

### Generate Hive DDL Commands

The SageMaker Python SDK's `FeatureStore` class also provides the functionality to generate Hive DDL commands. The schema of the table is generated based on the feature definitions. Columns are named after feature name and data-type are inferred based on feature type.

```
print(feature_group.as_hive_ddl())
```

Example output:

```
CREATE EXTERNAL TABLE IF NOT EXISTS sagemaker_featurestore.identity-feature-
group-27-19-33-00 (
    TransactionID INT
    id_01 FLOAT
    id_02 FLOAT
    id_03 FLOAT
    id_04 FLOAT
    ...
)
```

### Build a Training Dataset

Feature Store automatically builds an Amazon Glue data catalog when you create feature groups and you can turn this off if you want. The following describes how to create a single training dataset with feature values from both identity and transaction feature groups created earlier in this topic. Also, the following describes how to run an Amazon Athena query to join data stored in the offline store from both identity and transaction feature groups.

To start, create an Athena query using `athena_query()` for both identity and transaction feature groups. The `'table_name'` is the Amazon Glue table that is autogenerated by Feature Store.

```
identity_query = identity_feature_group.athena_query()
transaction_query = transaction_feature_group.athena_query()

identity_table = identity_query.table_name
transaction_table = transaction_query.table_name
```

## Write and Execute an Athena Query

You write your query using SQL on these feature groups, and then execute the query with the `.run()` command and specify your S3 bucket location for the data set to be saved there.

```
# Athena query
query_string = 'SELECT * FROM "'+transaction_table+'" LEFT JOIN "'+identity_table+'" ON
"' + transaction_table + '".transactionid = "' + identity_table + '".transactionid'

# run Athena query. The output is loaded to a Pandas dataframe.
dataset = pd.DataFrame()
identity_query.run(query_string=query_string,
    output_location='s3://'+default_s3_bucket_name+'/query_results/')
identity_query.wait()
dataset = identity_query.as_dataframe()
```

From here you can train a model using this data set and then perform inference.

## Delete a Feature Group

You can delete a feature group with the `delete` function.

```
feature_group.delete()
```

The following code example is from the fraud detection example.

```
identity_feature_group.delete()
transaction_feature_group.delete()
```

For more information, see the [Delete a feature group API](#)

## Adding required policies to your IAM role

To get started with Amazon SageMaker Feature Store you must add the required policy to your role, `AmazonSageMakerFeatureStoreAccess`. Below is a walkthrough on how to add it to your role through the console.

### Step 1: Access Amazon Management Console

Sign in to the Amazon Management Console and open the [IAM console](#).

### Step 2: Choose Roles

In the navigation pane on the left, choose **Roles**. The navigation pane is illustrated below with **Roles** outlined in red.

## Identity and Access Management (IAM)

---

### Dashboard

#### ▼ Access management

Groups

Users

Roles

Policies

Identity providers

Account settings

#### ▼ Access reports

Access analyzer

Archive rules

Analyzers

Settings

Credential report

Organization activity

Service control policies (SCPs)

## Step 3: Find your role

In the search bar, enter the role you are using for Amazon SageMaker Feature Store. Below illustrates with a red arrow where you enter your role.

### Roles

#### What are IAM roles?

IAM roles are a secure way to grant permissions to entities that you trust. Examples of entities include:

- IAM user in another account
- Application code running on an EC2 instance that needs to perform actions on other resources
- An AWS service that needs to act on resources in your account to provide its features
- Users from a corporate directory who use Identity federation with SAML

IAM roles issue keys that are valid for short durations, making them a more secure way to grant access.

#### Additional resources:

- [IAM Roles FAQ](#)
- [IAM Roles Documentation](#)
- [Tutorial: Setting Up Cross Account Access](#)
- [Common Scenarios for Roles](#)



## Step 4: Attach policy

After you find your role, choose **Attach policies**. Below illustrates this with a red arrow.

The screenshot shows the 'Permissions' tab selected in the top navigation bar. Below it, a section titled 'Permissions policies' is expanded. A callout box highlights the 'Get started with permissions' section, which states 'This role doesn't have any permissions yet. Get started by attaching one or more policies'. A blue 'Attach policies' button is visible, with a red arrow pointing towards it from the right.

Next you will enter in the search bar the required policy. Below illustrates the search bar you will use to enter the policy.

## Attach Permissions

The screenshot shows the 'Attach Permissions' interface. At the top left is a 'Create policy' button. Below it is a search bar with a magnifying glass icon and the placeholder text 'Search'. To the left of the search bar is a 'Filter policies' dropdown menu. Below these are two columns: 'Policy name' and a status column. A red arrow points from the right side of the image towards the search bar.

The policy you will need to add is `AmazonSageMakerFeatureStoreAccess`. After you enter the policy, select the **check box** and then choose **Attach policy**. Below illustrates the this step.

## Attach Permissions

Create policy

The screenshot shows the AWS IAM Policies page. At the top left is a 'Create policy' button. To its right is a search bar with the placeholder text 'AmazonSageMakerFeatureStoreAccess'. Below the search bar is a 'Filter policies' dropdown menu. The main area displays a table with two columns: 'Policy name' and 'Status'. The first row in the table is highlighted with a red box around the checkbox column. The second row contains the policy name 'AmazonSageMakerFeatureStoreAccess' and a checked checkbox, also highlighted with a red box.

Policy name	Status
AmazonSageMakerFeatureStoreAccess	<input checked="" type="checkbox"/>

After you have attached both policies to your role, the policy should appear under your IAM role. The following illustrates how it appears under your IAM role.

The screenshot shows the 'Permissions' tab selected in the top navigation bar. Below it, a section titled 'Permissions policies (1 policy applied)' is expanded, showing a single policy named 'AmazonSageMakerFeatureStoreAccess'. A blue button labeled 'Attach policies' is visible above the policy list. The policy name is highlighted with a red box.

▼ Permissions policies (1 policy applied)

Attach policies

Policy name ▾

▶ **AmazonSageMakerFeatureStoreAccess**

▶ Permissions boundary (not set)

## Use Amazon SageMaker Feature Store with Amazon SageMaker Studio

You can use Studio to create and view details about your feature groups.

### Topics

- [Create a Feature Group in Studio \(p. 691\)](#)
- [View Feature Group Details in Studio \(p. 696\)](#)

## Create a Feature Group in Studio

The create feature group process in Studio has four steps: enter details, definitions, required features, and tags.

Consider the following options before you start:

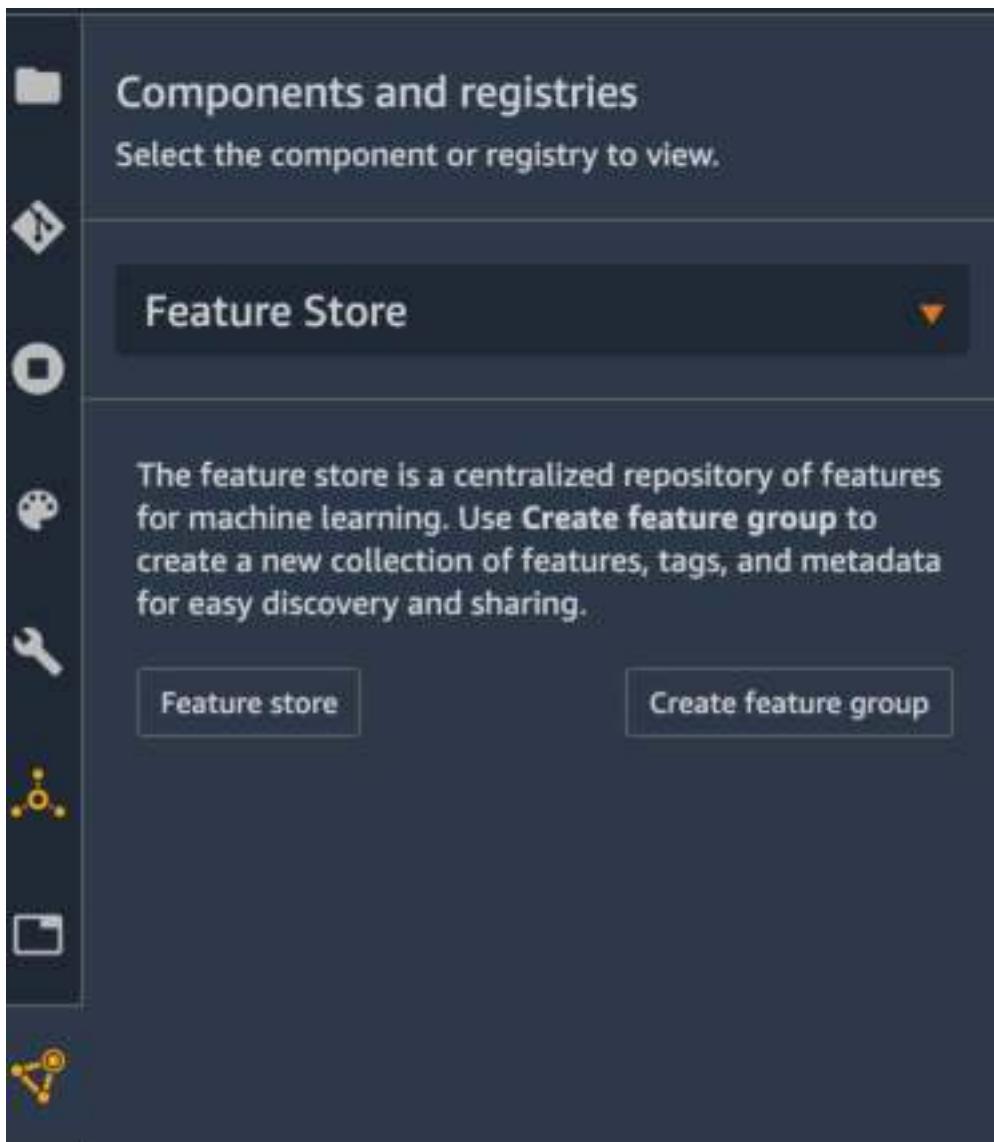
- If you plan to only use an online store, you just need the schema for your features. This is your columns and each column's data type.
- If you plan to use an offline store, you need an S3 bucket URI and a Role ARN.
- If you plan to use encryption, you need a KMS key. You can use the same one for both the online store and your offline store, or have a unique key for each.
- If you plan to use Amazon Glue integration, be prepared to provide a data catalog name, database name, and table name.

### To create a feature group in Studio

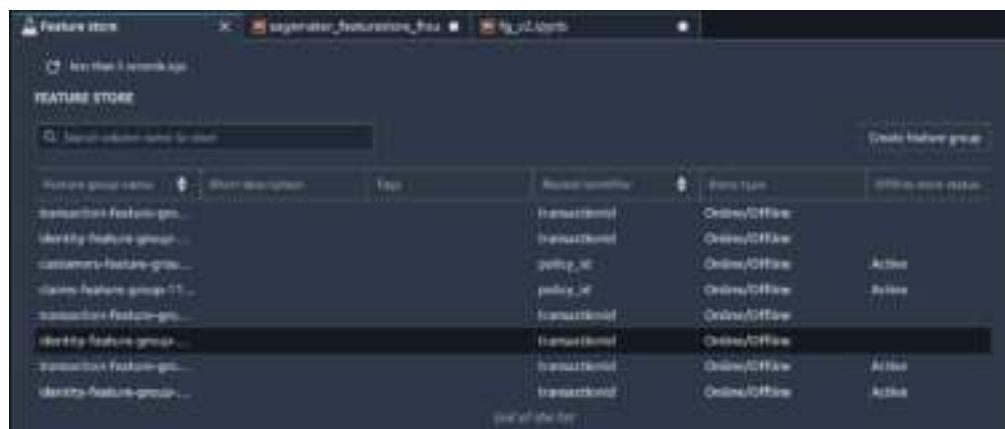
1. Sign in to Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
2. In the left navigation pane, choose the **Components and registries** icon (  ).



3. In the file and resource browser, choose **Feature Store**.



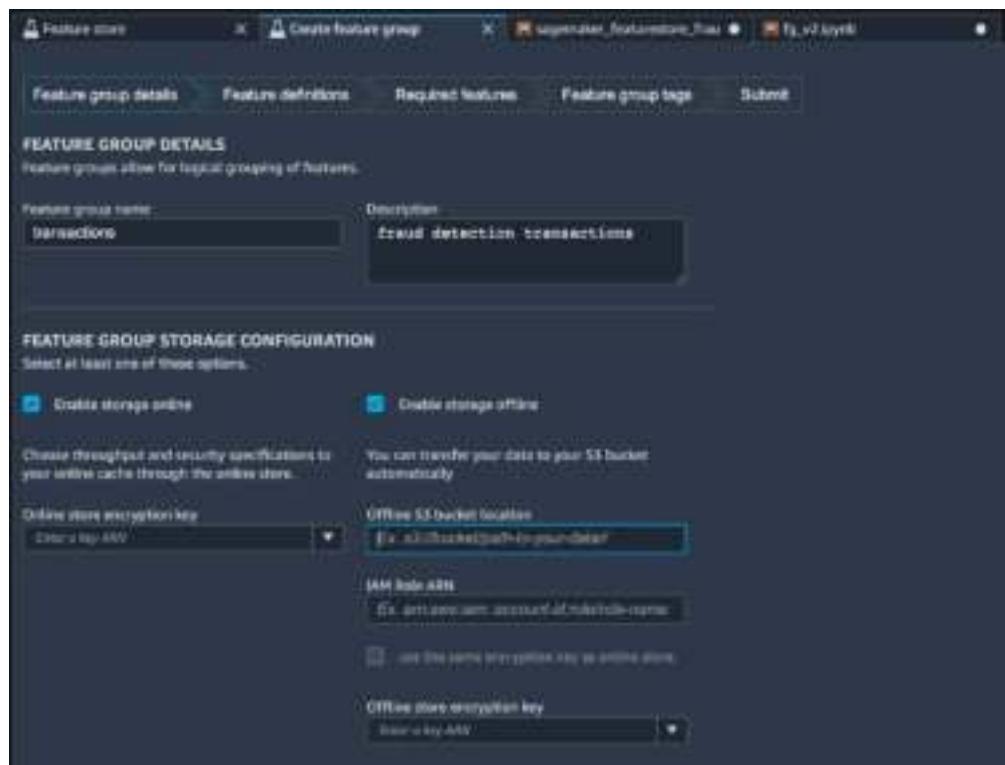
4. The **Feature Store** tab lists your feature groups.



The screenshot shows a table listing feature groups within a feature store. The columns are: Feature group name, Description, Type, Access control, Data type, and Status.

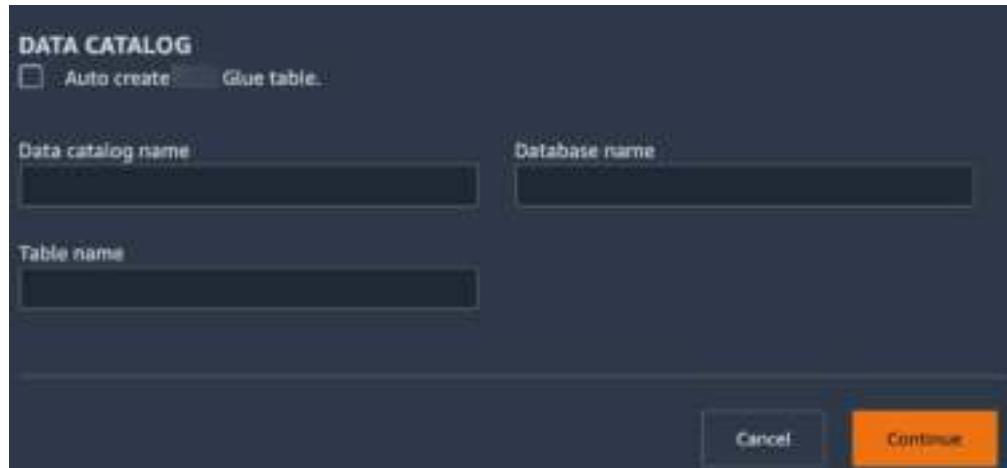
Feature group name	Description	Type	Access control	Data type	Status
transaction-feature-group		transactional	Online/Offline	Online	Active
identity-feature-group		transactional	Online/Offline	Online	Active
customer-feature-group		offline	Online/Offline	Active	Active
claims-feature-group-01		offline	Online/Offline	Online	Active
transaction-feature-group		transactional	Online/Offline	Online	Active
device-feature-group		transactional	Online/Offline	Online	Active
customer-feature-group-02		transactional	Online/Offline	Online	Active
device-feature-group-01		transactional	Online/Offline	Online	Active

5. The **Feature Store** tab, choose **Create Feature Group**.
6. Enter the feature group details.

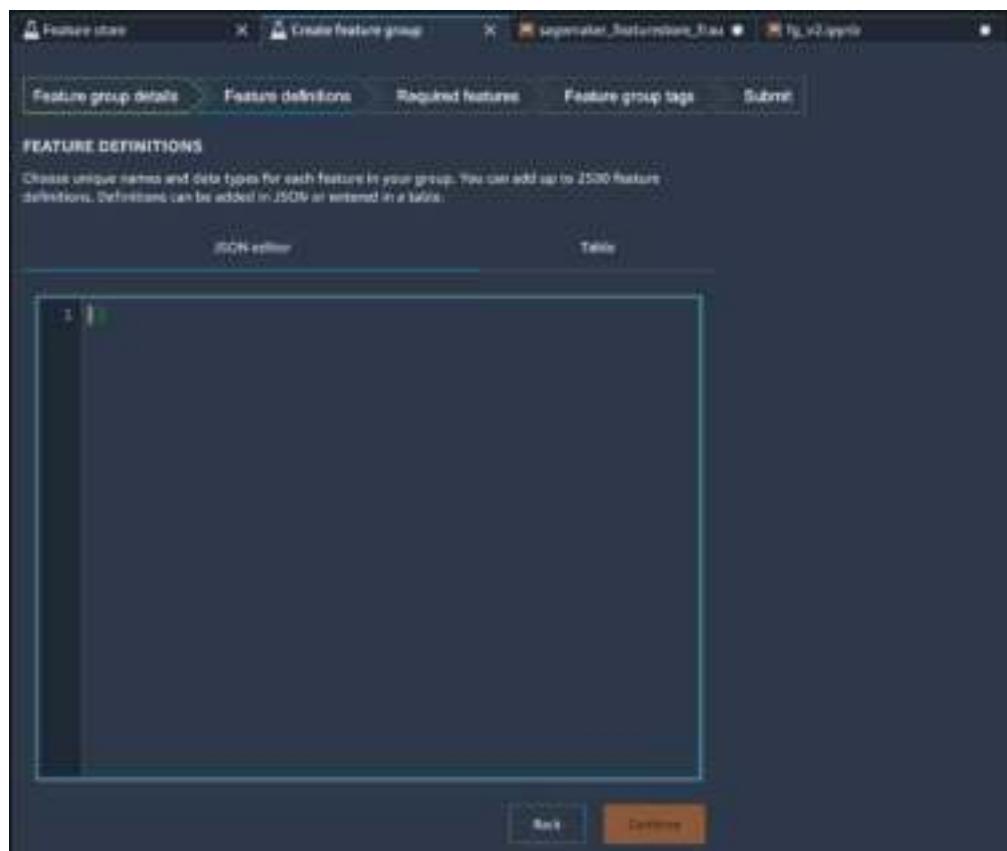


The screenshot shows the 'Create feature group' wizard with the 'Feature group details' step selected. The form includes fields for feature group name ('transactions') and description ('credit detection transactions'). It also includes sections for 'FEATURE GROUP STORAGE CONFIGURATION' (with 'Enable storage online' checked), 'AWS Lambda ARN' (with 'arn:aws:lambda:us-east-1:123456789012:function:my-lambda-function' entered), and 'Offline store encryption key' (with 'External ARN' dropdown set to 'External ARN').

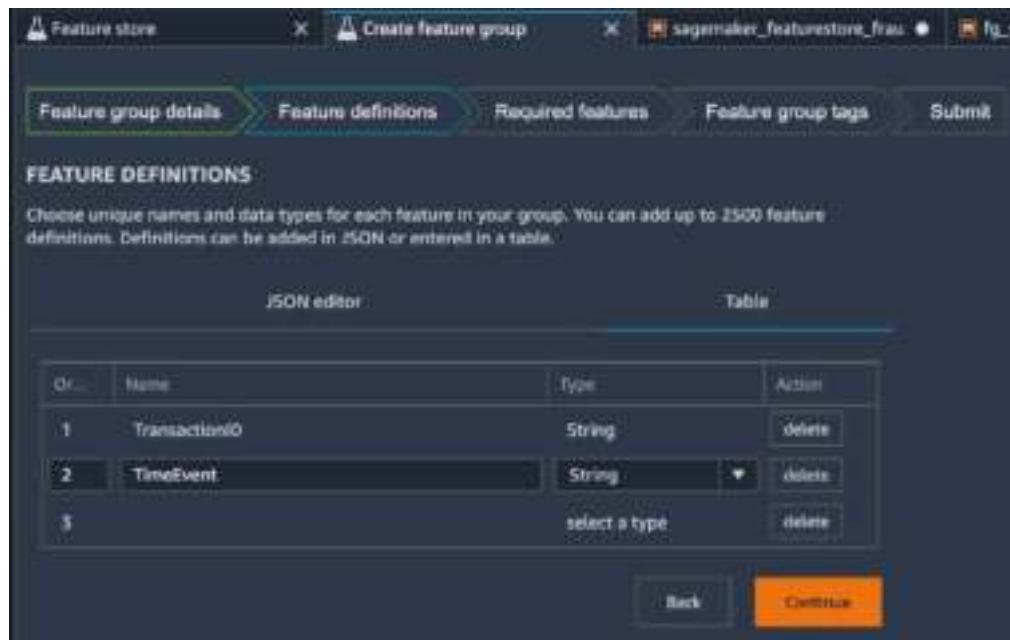
7. (Optional) If you're using Glue, enter the data catalog details.



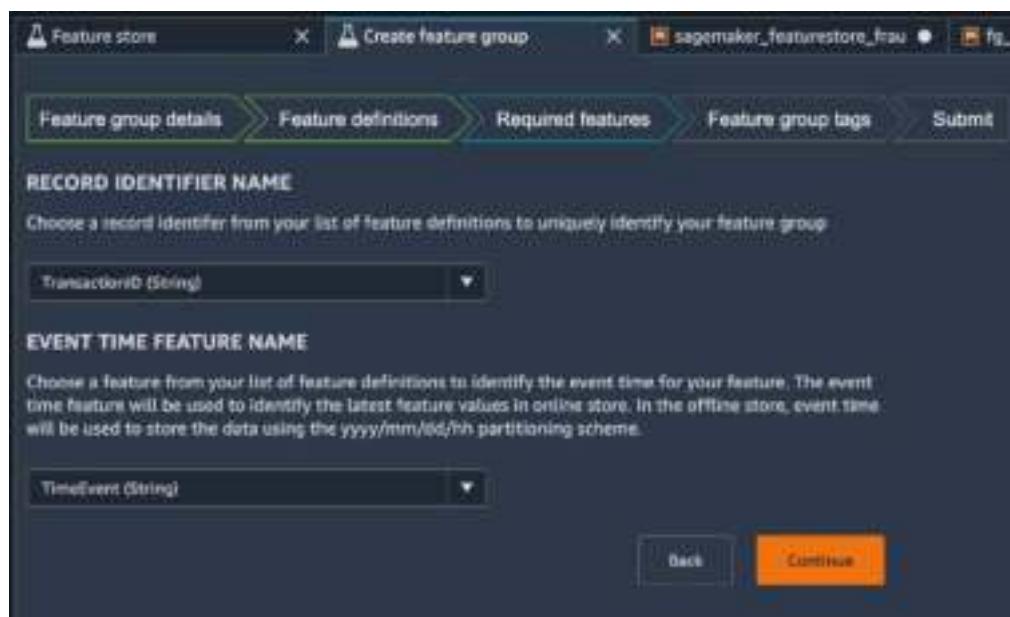
8. Enter feature definitions. You have two options for providing a schema for your features: a JSON editor, or a Table editor.



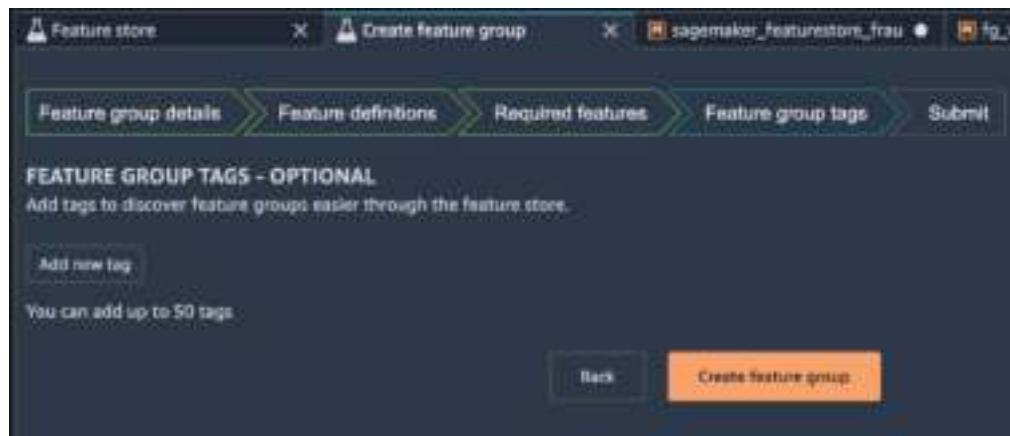
9. The table editor accepts a column name and a data type. In a minimal example, you will need at least two for the next step: one for a record identifier and one for a time event feature. You can have up to 2,500 feature definitions.



10. Set a record identifier and a time feature to use for this feature group.



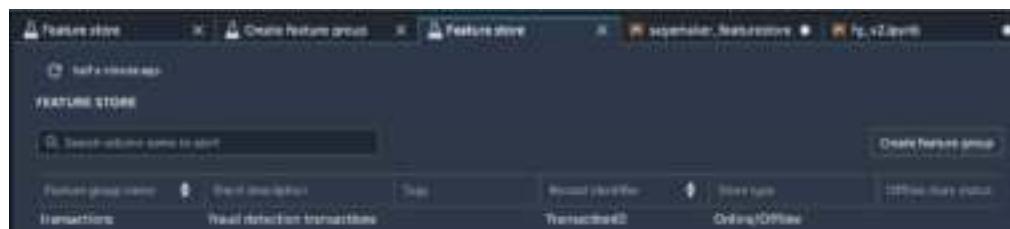
11. (Optional) Enter tags as key value pairs.



12. Choose **Create feature group**.



13. In the **Actions** column, choose **Open feature store**.



14. When the feature group is finished being created, it appears in your feature groups list. Choose the refresh button to refresh the list.

## View Feature Group Details in Studio

You can view details about your feature groups, and get sample queries to run against your data sources to gather the data for features you have defined.

### To view feature group details in Studio

1. Double-click or right-click a feature group from your list, and then choose **Open feature group detail**.

FEATURE STORE		
<input type="text"/> Search column name to start		
Feature group name	Short description	Tags
transactions	fraud detection transactions	
transaction-feature	Open feature group detail	
identity-feature-g	Copy cell contents	
customers-feature	Shift+Right Click for Browser Menu	
claims-feature-grou...		

2. In the details view, you can review your feature group summary, feature definitions, feature group tags, and sample query.

FEATURE GROUP: transactions			
Feature group summary	Feature definitions	Feature group tags	Sample query
<b>Feature group details</b>			
Feature group name:	Description:		
transactions	fraud detection transactions		
Feature group ARN:	Record identifier:		
arn:aws:sagemaker:us-east-2:678128118188:feature-group/transactions	TransactionID		
	Created on:		
	2020-11-29T15:19:14.000Z		
<b>Feature group status</b>			
Online store status:			
Created:			
<b>Feature group storage configuration</b>			
Enable online storage:	Enable offline storage:		
Enabled	arn:aws:iam:678128118188:role/DataprepRole		
	s3://sagemaker-featurestore-678128118188		

3. On the **Feature definitions** tab, you can search for features by name.

The screenshot shows the 'FEATURE GROUP: transactions' page. At the top, there are four tabs: 'Feature group summary' (selected), 'Feature definitions', 'Feature group tags', and 'Sample query'. Below the tabs is a search bar labeled 'Search by feature name'. A table follows, with columns 'Feature name' and 'Feature Type'. It contains two rows: 'TransactionID' (String) and 'TimeEvent' (String).

4. On the **Feature group tags** tab, you can add and remove tags.

The screenshot shows the 'FEATURE GROUP: transactions' page with the 'Feature group tags' tab selected. A message at the top says 'Add or remove feature group tags. You can add up to 50 feature group tags.' Below is a button labeled 'Add new tag'.

5. On the **Sample query** tab, you can see a variety of sample queries: interactive exploration, time travel, remove tombstone, and remove duplicates.

The screenshot shows the 'FEATURE GROUP: transactions' page with the 'Sample query' tab selected. A message at the top says 'Use the buttons below to generate the query to perform the action with the feature data for this feature group. You can copy and paste this query to use with SageMaker Data Wrangler or in to any query interface for querying data from the offline store.' Below are four buttons: 'Interactive Exploration' (selected), 'Time travel.', 'Remove tombstone.', and 'Remove duplicates.'. A code editor at the bottom contains a SQL query:

```
SELECT *
FROM <offlineStoreConfig.dataCatalogConfig.database>,
<offlineStoreConfig.dataCatalogConfig.tableName>
LIMIT 1000
```

## Data Sources and Ingestion

There are multiple ways to bring your data into Amazon SageMaker Feature Store. Feature Store offers a single API call for data ingestion called `PutRecord` that enables you to ingest data in batches or from

streaming sources. You can also use Amazon SageMaker Data Wrangler to engineer features and then ingest your features into your Feature Store.

#### Topics

- [Stream Ingestion \(p. 699\)](#)
- [Data Wrangler with Feature Store \(p. 699\)](#)
- [Athena and Amazon Glue with Feature Store \(p. 701\)](#)

## Stream Ingestion

You can use streaming sources such as Kafka or Kinesis as a data source where features are extracted from there and directly fed to the online feature store for training, inference or feature creation. Records can be pushed into the feature store by calling the synchronous `PutRecord` API call. Since this is a synchronous API call it allows small batches of updates to be pushed in a single API call. This enables you to maintain high freshness of the feature values and publish values as soon an update is detected. These are also called *streaming* features.

## Data Wrangler with Feature Store

Data Wrangler is a feature of Studio that provides an end-to-end solution to import, prepare, transform, featurize, and analyze data. Data Wrangler enables you to engineer your features and ingest them into a feature store.

In Studio, after interacting with Data Wrangler, choose the **Export** tab, choose **Export Step**, and the choose **Feature Store**, as shown in the following screenshot. This exports a Jupyter notebook that has all the source code in it to create a Feature Store feature group that adds yours features from Data Wrangler to an offline or online feature store.



After the feature group has been created, you can also select and join data across multiple feature groups to create new engineered features in Data Wrangler and then export your data set to an S3 bucket.

For more information on how to export to Feature Store, see [Export to SageMaker Feature Store](#).

## Athena and Amazon Glue with Feature Store

After a Feature Store feature group has been created in an offline feature store, you can choose to run queries using Amazon Athena on a Amazon Glue catalog. This requires data to be registered in a data catalog with other catalog details which is auto-registered for you in Feature Store. In other words, Feature Store automatically builds an Amazon Glue data catalog when feature groups are created and you can turn them off. This is particularly useful when you want to build a dataset by executing SQL queries and then train a model for inference.

After your `FeatureStore` has been created and populated with your data in the offline store, you have the capability to write SQL queries to join data stored in the offline store from different `FeatureGroups`. To do this, you can use Amazon Athena to write and execute SQL queries. You can set up a Amazon Glue crawler to run on a schedule to ensure your catalog is always up to date as well.

If you want to do this please define a role which can be used by the Amazon Glue crawler to access the offline store's S3 buckets. For more information, see [Create an IAM role](#).

For more information on how to use Amazon Glue and Athena to build a training dataset for model training and inference, see [Build Training Dataset: Create Feature Groups](#).

## Sample Athena Queries

Below we provide some sample queries that act as a template for you to quickly write queries using Athena.

### Interactive Exploration

This query selects the first 1000 records.

```
SELECT *
FROM <FeatureGroup.DataCatalogConfig.DatabaseName>.<FeatureGroup.DataCatalogConfig.TableName>
LIMIT 1000
```

### Latest snapshot without duplicates

This query selects the latest non-duplicate records.

```
SELECT *
FROM
  (SELECT *,
    row_number()
  OVER (PARTITION BY <RecordIdentifierFeatureName>
  ORDER BY <EventTimeFeatureName> desc, Api_Invocation_Time DESC, write_time DESC) AS
  row_num
  FROM
    <FeatureGroup.DataCatalogConfig.DatabaseName>.<FeatureGroup.DataCatalogConfig.TableName>
  WHERE row_num = 1;
```

### Latest snapshot without duplicates and deleted records in the offline store

This query filters out any deleted records and selects non-duplicate records from the offline store.

```
SELECT *
FROM
  (SELECT *,
```

```

        row_number()
    OVER (PARTITION BY <RecordIdentifierFeatureName>
    ORDER BY <EventTimeFeatureName> desc, Api_Invocation_Time DESC, write_time DESC) AS
row_num
    FROM
<FeatureGroup.DataCatalogConfig.DatabaseName>.<FeatureGroup.DataCatalogConfig.TableName>
WHERE row_num = 1 and
NOT is_deleted;

```

#### Time Travel without duplicates and deleted records in the offline store

This query filters out any deleted records and selects non-duplicate records from a particular point in time.

```

SELECT *
FROM
(SELECT *, 
    row_number()
OVER (PARTITION BY <RecordIdentifierFeatureName>
    ORDER BY <EventTimeFeatureName> desc, Api_Invocation_Time DESC, write_time DESC) AS
row_num
    FROM
<FeatureGroup.DataCatalogConfig.DatabaseName>.<FeatureGroup.DataCatalogConfig.TableName>
    where <EventTimeFeatureName> <= timestamp '<timestamp>')
    -- replace timestamp '<timestamp>' with just <timestamp> if EventTimeFeature is of
    type fractional
WHERE row_num = 1 and
NOT is_deleted

```

## Security and Access Control

Amazon SageMaker Feature Store enables you to create two types of stores: an online store or offline store. The online store is used for low latency real-time inference use cases whereas the offline store is used for training and batch inference use cases. When you create a feature group for online or offline use you can provide a KMS customer master key (CMK) to encrypt all your data at rest. In case you do not provide a CMK then we ensure that your data is encrypted on the server side using an Amazon owned CMK or Amazon managed CMK. While creating a feature group, you can select storage type and optionally provide a CMK for encrypting data, then you can call various APIs for data management such as `PutRecord`, `GetRecord`, `DeleteRecord`.

Feature Store allows you to grant or deny access to individuals at the feature group-level and enables cross-account access to Feature Store. For example, you can set up developer accounts to access the offline store for model training and exploration that do not have write access to production accounts. You can set up production accounts to access both online and offline stores. Feature Store uses unique customer Amazon KMS CMKs for offline and online store data at-rest encryption. Access control is enabled through both API and KMS CMK access. You can also create feature group-level access control.

For more information about CMKs, see [Customer Managed Keys](#). For more information about KMS, please see [KMS](#).

## Using KMS Permissions for Amazon SageMaker Feature Store

Encryption at rest protects Feature Store under an Amazon KMS customer master key (CMK). By default, it uses an [Amazon owned CMK for OnlineStore](#) and [Amazon managed CMK for OfflineStore](#). Feature Store supports an option to encrypt your online or offline store under [customer managed CMKs](#). You can

select the CMK for Feature Store when you create your online or offline store, and they can be different for each store.

Feature Store supports only [symmetric CMKs](#). You cannot use an [asymmetric CMK](#) to encrypt your data in your online or offline store. For help determining whether a CMK is symmetric or asymmetric, see [Identifying symmetric and asymmetric CMKs](#).

When you use a customer managed CMK, you can take advantage of the following features:

- You create and manage the CMK, including setting the [key policies](#), [IAM policies](#) and [grants](#) to control access to the CMK. You can [enable and disable](#) the CMK, enable and disable [automatic key rotation](#), and [delete the CMK](#) when it is no longer in use.
- You can use a customer managed CMK with [imported key material](#) or a customer managed CMK in a [custom key store](#) that you own and manage.
- You can audit the encryption and decryption of your online or offline store by examining the API calls to Amazon KMS in [Amazon CloudTrail logs](#).

You do not pay a monthly fee for Amazon owned CMKs. Customer managed CMKs [incur a charge](#) for each API call and Amazon KMS quotas apply to these CMKs.

## Authorizing Use of Your CMK for Your Online Store

If you use a [customer managed CMK](#) to protect your online store, the policies on that CMK must give Feature Store permission to use it on your behalf. You have full control over the policies and grants on a customer managed CMK.

Feature Store does not need additional authorization to use the default [Amazon owned CMK](#) to protect your online or offline stores in your Amazon account.

### Customer Managed CMK Key Policy

When you select a [customer managed CMK](#) to protect your Online Store, Feature Store must have permission to use the CMK on behalf of the principal who makes the selection. That principal, a user or role, must have the permissions on the CMK that Feature Store requires. You can provide these permissions in a [key policy](#), an [IAM policy](#), or a [grant](#). At a minimum, Feature Store requires the following permissions on a customer managed CMK:

- "kms:Encrypt", "kms:Decrypt", "kms:DescribeKey", "kms>CreateGrant", "kms:RetireGrant", "kms:ReEncryptFrom", "kms:ReEncryptTo", "kms:GenerateDataKey", "kms>ListAliases", "kms>ListGrants", "kms:RevokeGrant"

For example, the following example key policy provides only the required permissions. The policy has the following effects:

- Allows Feature Store to use the CMK in cryptographic operations and create grants, but only when it is acting on behalf of principals in the account who have permission to use your Feature Store. If the principals specified in the policy statement don't have permission to use your Feature Store, the call fails, even when it comes from the Feature Store service.
- The [kms:ViaService](#) condition key allows the permissions only when the request comes from FeatureStore on behalf of the principals listed in the policy statement. These principals can't call these operations directly. The value for `kms:ViaService` should be `sagemaker.*.amazonaws.com`.

#### Note

The `kms:ViaService` condition key can only be used for the online store customer managed KMS key, and cannot be used for the offline store. If you add this special condition to your customer managed key, and use the same KMS key for both the online and offline store, then it will fail the `CreateFeatureGroup` API operation.

- Gives the CMK administrators read-only access to the CMK and permission to revoke grants, including the grants that Feature Store uses to protect your data.
- Gives Feature Store read-only access to the CMK. In this case, Feature Store can call these operations directly. It does not have to act on behalf of an account principal.

Before using an example key policy, replace the example principals with actual principals from your Amazon account.

```
{
  "Id": "key-policy-feature-store",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Allow access through Amazon SageMaker Feature Store for all principals in the account that are authorized to use Amazon SageMaker Feature Store",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:user/featurestore-user"},
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:DescribeKey",
        "kms>CreateGrant",
        "kms:RetireGrant",
        "kms:ReEncryptFrom",
        "kms:ReEncryptTo",
        "kms:GenerateDataKey",
        "kms>ListAliases",
        "kms>ListGrants"
      ],
      "Resource": "*",
      "Condition": {"StringLike": {"kms:ViaService": "sagemaker.*.amazonaws.com"}}
    },
    {
      "Sid": "Allow administrators to view the CMK and revoke grants",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:role/featurestore-admin"},
      "Action": [
        "kms:Describe*",
        "kms:Get*",
        "kms>List*",
        "kms:RevokeGrant"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Allow Feature Store to get information about the CMK",
      "Effect": "Allow",
      "Principal": {"Service": ["sagemaker.amazonaws.com"]},
      "Action": [
        "kms:Describe*",
        "kms:Get*",
        "kms>List*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "Enable IAM User Permissions",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::123456789:root"},
      "Action": "kms:*",
      "Resource": "*"
    }
  ]
}
```

}

## Using Grants to Authorize Feature Store

In addition to key policies, Feature Store uses grants to set permissions on the customer managed CMK. To view the grants on a CMK in your account, use the [ListGrants](#) operation. Feature Store does not need grants, or any additional permissions, to use the [Amazon owned CMK](#) to protect your online store.

Feature Store uses the grant permissions when it performs background system maintenance and continuous data protection tasks.

Each grant is specific to an online store. If the account includes multiple stores encrypted under the same CMK, there will be unique grants per `FeatureGroup` using the same CMK.

The key policy can also allow the account to [revoke the grant](#) on the CMK. However, if you revoke the grant on an active encrypted online store, Feature Store won't be able to protect and maintain the store.

## Monitoring Feature Store interaction with Amazon KMS

If you use a [customer managed CMK](#) to protect your online or offline store, you can use Amazon CloudTrail logs to track the requests that Feature Store sends to Amazon KMS on your behalf.

## Accessing Data in Your Online Store

The **caller (either IAM user or IAM role)** to **ALL DataPlane operations (Put, Get, DeleteRecord)** must have below permissions on the customer managed CMK:

"`kms:Decrypt`"

## Authorizing Use of Your CMK for Offline Store

The **roleArn** that is passed as a parameter to `createFeatureGroup` must have below permissions to the OfflineStore KmsKeyId:

"`kms:GenerateDataKey`"

### Note

The key policy for the online store also works for the offline store, only when the `kms:ViaService` condition is not specified.

## Cross-Account Offline Store Access

Amazon SageMaker Feature Store allows users to create a feature group in one account (Account A) and configure it with an offline store using an Amazon S3 bucket in another account (Account B). This can be set up using the steps in the following section.

### Topics

- [Step 1: Set Up the Offline Store Access Role in Account A \(p. 706\)](#)
- [Step 2: Set up an Offline Store S3 Bucket in Account B \(p. 707\)](#)
- [Step 3: Set up an Offline Store KMS Encryption Key in Account A \(p. 707\)](#)

- [Step 4: Create a Feature Group in Account A \(p. 709\)](#)

## Step 1: Set Up the Offline Store Access Role in Account A

First, set up a role for Amazon SageMaker Feature Store to write the data into the offline store. The simplest way to accomplish this is to create a new role using the `AmazonSageMakerFeatureStoreAccess` policy or to use an existing role that already has the `AmazonSageMakerFeatureStoreAccess` policy attached. This document refers to this policy as `Account-A-Offline-Feature-Store-Role-ARN`.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutObject",  
                "s3:GetBucketAcl",  
                "s3:PutObjectAcl"  
            ],  
            "Resource": [  
                "arn:aws:s3:::*SageMaker*",  
                "arn:aws:s3:::*Sagemaker*",  
                "arn:aws:s3:::*sagemaker*"  
            ]  
        }  
    ]  
}
```

The preceding code snippet shows the `AmazonSageMakerFeatureStoreAccess` policy. The `Resource` section of the policy is scoped down by default to S3 buckets with names that contain `SageMaker`, `Sagemaker`, or `sagemaker`. This means the offline store S3 bucket being used must follow this naming convention. If this is not your case, or if you want to further scope down the resource, you can copy and paste the policy to your S3 bucket policy in the console, customize the `Resource` section to be `arn:aws:s3:::your-offline-store-bucket-name`, and then attach to the role.

Additionally, this role must have KMS permissions attached. At a minimum, it requires the `kms:GenerateDataKey` permission to be able to write to the offline store using your customer managed CMK. See Step 3 to learn about why a customer managed CMK is needed for the cross-account scenario and how to set it up. The following example shows an inline policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "VisualEditor0",  
            "Effect": "Allow",  
            "Action": [  
                "kms:GenerateDataKey"  
            ],  
            "Resource": "arn:aws:kms::Account-A-Account-Id:key/*"  
        }  
    ]  
}
```

The `Resource` section of this policy is scoped to any key in Account A. To further scope this down, after setting up the offline store KMS key in Step 3, return to this policy and replace it with the key ARN.

## Step 2: Set up an Offline Store S3 Bucket in Account B

Create an S3 bucket in Account B. If you are using the default `AmazonSageMakerFeatureStoreAccess` policy, the bucket name must include `SageMaker`, `Sagemaker`, or `sagemaker`. Edit the bucket policy as shown in the following example to allow Account A to read and write objects.

This document refers to the following example bucket policy as `Account-B-Offline-Feature-Store-Bucket`.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "S3CrossAccountBucketAccess",  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutObject",  
                "s3:PutObjectAcl",  
                "s3:GetBucketAcl"  
            ],  
            "Principal": {  
                "AWS": [  
                    "*Account-A-Offline-Feature-Store-Role-ARN*"  
                ],  
            },  
            "Resource": [  
                "arn:aws:s3:::offline-store-bucket-name/*",  
                "arn:aws:s3:::offline-store-bucket-name"  
            ]  
        }  
    ]  
}
```

In the preceding policy, the principal is `Account-A-Offline-Feature-Store-Role-ARN`, which is the role created in Account A in Step 1 and provided to Amazon SageMaker Feature Store to write to the offline store. You can provide multiple ARN roles under `Principal`.

## Step 3: Set up an Offline Store KMS Encryption Key in Account A

Amazon SageMaker Feature Store ensures that server-side encryption is always enabled for S3 objects in the offline store. For cross-account use cases, you must provide a customer managed CMK so that you are in control of who can write to the offline store (in this case, `Account-A-Offline-Feature-Store-Role-ARN` from Account A) and who can read from the offline store (in this case, identities from Account B).

This document refers to the following example key policy as `Account-A-Offline-Feature-Store-KMS-Key-ARN`.

```
{  
    "Version": "2012-10-17",  
    "Id": "key-consolepolicy-3",  
    "Statement": [  
        {  
            "Sid": "Enable IAM User Permissions",  
            "Effect": "Allow",  
            "Principal": "AWS",  
            "Action": "kms:CreateKey",  
            "Resource": "  
                "arn:aws:kms:  
                    <region>:  
                    <account>/key/  
                    <key-id>"  
            "Condition": {  
                "StringEquals": {  
                    "AWS:SourceIdentity": "AWS",  
                    "AWS:SourceArn": "  
                        arn:aws:iam:  
                            <region>:  
                            <account>/role/  
                            <role-name>"  
                }  
            }  
        }  
    ]  
}
```

```
"Effect": "Allow",
"Principal": {
    "AWS": "arn:aws:iam::Account-A-Account-Id:root"
},
"Action": "kms:*",
"Resource": "*"
},
{
    "Sid": "Allow access for Key Administrators",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "arn:aws:iam::Account-A-Account-Id:role/Administrator",
        ]
    },
    "Action": [
        "kms>Create*",
        "kms>Describe*",
        "kms>Enable*",
        "kms>List*",
        "kms>Put*",
        "kms>Update*",
        "kms>Revoke*",
        "kms>Disable*",
        "kms>Get*",
        "kms>Delete*",
        "kms>TagResource",
        "kms>UntagResource",
        "kms>ScheduleKeyDeletion",
        "kms>CancelKeyDeletion"
    ],
    "Resource": "*"
},
{
    "Sid": "Allow Feature Store to get information about the CMK",
    "Effect": "Allow",
    "Principal": {
        "Service": "sagemaker.amazonaws.com"
    },
    "Action": [
        "kms>Describe*",
        "kms>Get*",
        "kms>List*"
    ],
    "Resource": "*"
},
{
    "Sid": "Allow use of the key",
    "Effect": "Allow",
    "Principal": {
        "AWS": [
            "*Account-A-Offline-Feature-Store-Role-ARN*",
            "*arn:aws:iam::Account-B-Account-Id:root*"
        ]
    },
    "Action": [
        "kms>Encrypt",
        "kms>Decrypt",
        "kms>DescribeKey",
        "kms>CreateGrant",
        "kms>RetireGrant",
        "kms>ReEncryptFrom",
        "kms>ReEncryptTo",
        "kms>GenerateDataKey",
        "kms>ListAliases",
        "kms>ListGrants"
```

```
        ],
        "Resource": "*",
    }
}
```

## Step 4: Create a Feature Group in Account A

Next, create the feature group in Account A, with an offline store S3 bucket in Account B. To do this, provide the following parameters for `RoleArn`, `OfflineStoreConfig.S3StorageConfig.KmsKeyId` and `OfflineStoreConfig.S3StorageConfig.S3Uri` respectively:

- Provide `Account-A-Offline-Feature-Store-Role-ARN` as the `RoleArn`.
- Provide `Account-A-Offline-Feature-Store-KMS-Key-ARN` for `OfflineStoreConfig.S3StorageConfig.KmsKeyId`.
- Provide `Account-B-Offline-Feature-Store-Bucket` for `OfflineStoreConfig.S3StorageConfig.S3Uri`.

# Quotas, Naming Rules and Data Types

## Limits and Quotas

### Note

Soft limits can be increased based on your needs.

- **Maximum number of feature groups per Amazon account:** Soft limit of 100.
- **Maximum number of feature definitions per feature group:** 2500.
- **Maximum Transactions per second (TPS) per API per Amazon account:** Soft limit of 10000 TPS per API.
- **Maximum size of a record:** 350KB.
- **Maximum size of a feature value:** 350KB.
- **Maximum number of concurrent feature group creation workflows:** 4.

## Naming Rules

- **Reserved Words:** The following are reserved words and cannot be used as feature names in feature definitions: `is_deleted`, `write_time`, and `api_invocation_time`.

## Data Types

- **String Feature Type:** Strings are Unicode with UTF-8 binary encoding. The minimum length of a string can be zero, the maximum length is constrained by the maximum size of a record.
- **Fractional Feature Type:** Fractional feature values must conform to a double precision floating point number as defined by the [IEEE 754 standard](#).
- **Integral Feature Type:** Feature Store supports integral values in the range of a 64-bit signed integer. Minimum value of  $-2^{63}$  and a maximum value:  $2^{63} - 1$ .
- **Event Time Features:** All feature groups have an event time feature. The feature can have a feature type of either String or Fractional. A string event time is accepted in ISO-8601 format, in UTC time, conforming to the pattern(s): [yyyy-MM-dd'T'HH:mm:ssZ, yyyy-MM-dd'T'HH:mm:ss.SSSZ]. A fractional

event time value is accepted as seconds from unix epoch, with millisecond precision. Event times must be in the range of [0000-01-01T00:00:00.000Z, 9999-12-31T23:59:59.999Z].

## Amazon SageMaker Feature Store Offline Store Data Format

Amazon SageMaker Feature Store offline store data is stored in an Amazon S3 bucket within your account. When you call `PutRecord`, your data is buffered, batched, and written into Amazon S3 within 15 minutes. Feature Store only supports the Parquet file format. Specifically, when your data is written to your offline store, the data can only be retrieved from your Amazon S3 bucket in Parquet format. Each file can contain multiple Records.

Files are organized with the following naming convention:

```
s3://<bucket-name>/<customer-prefix>/<account-id>/sagemaker/<aws-region>/offline-store/
<feature-group-name>-<feature-group-creation-time>/data/year=<event-time-year>/
    month=<event-time-month>/day=<event-time-day>/hour=<event-time-hour>/
<timestamp_of_latest_event_time_in_file>_<16-random-alphanumeric-digits>.parquet
```

For example:

```
s3://<my-bucket/my-prefix/123456789012/sagemaker/us-east-1/offline-store/
    customer-purchase-history-patterns-1593511200/data/year=2020/month=06/day=31/
    hour=00/20130917T064401Z_108934320012Az11.parquet
```

Feature Store also exposes the `OfflineStoreConfig.S3StorageConfig.ResolvedOutputS3Uri` field, which can be found from in the [DescribeFeatureGroup](#) API call. This is the S3 path under which the files for the specific feature group are written.

Example value of `ResolvedOutputS3Uri`:

```
s3://<my-bucket/my-prefix/123456789012/sagemaker/us-east-1/offline-store/customer-purchase-
history-patterns-1593511200/data
```

The following additional fields are added by Feature Store to each Record when they persist in the offline store:

- **api\_invocation\_time** – The timestamp when the service receives the `PutRecord` or `DeleteRecord` call. If using managed ingestion (e.g. Data Wrangler), this is the timestamp when data was written into the offline store.
- **write\_time** – The timestamp when data was written into the offline store. Can be used for constructing time-travel related queries.
- **is\_deleted** – `False` by default. If `DeleteRecord` is called, a new Record is inserted into `RecordIdentifierValue` and set to `True` in the offline store.

## Amazon SageMaker Feature Store Notebook Examples

To get started using Amazon SageMaker Feature Store, you can choose from a variety of example Jupyter notebooks in the table below. If this is your first time using Feature Store, try out the

Introduction to Feature Store notebook. To run any of these notebooks, you must attach this policy to your IAM execution role: `AmazonSageMakerFeatureStoreAccess`.

See [IAM Roles](#) to access your role and attach this policy. For a walkthrough see [Adding policies to your IAM role](#). The following screenshot illustrates how the policy appears under your IAM role after it has been attached.

The screenshot shows the 'Permissions' tab of an IAM role configuration. At the top, there are tabs for 'Permissions', 'Trust relationships', 'Tags', and 'Access Advisor'. Below these tabs, a section titled 'Permissions policies (1 policy applied)' is expanded, showing a blue button labeled 'Attach policies'. A red box highlights the 'Policy name' dropdown, which contains a single item: 'AmazonSageMakerFeatureStoreAccess' preceded by an orange cube icon. Below this, another section titled 'Permissions boundary (not set)' is shown.

## Feature Store sample notebooks

The following table outlines a variety of sample notebooks that address different use cases of Amazon SageMaker Feature Store.

Notebook Title	Description
<a href="#">Introduction to Feature Store</a>	An introduction to key Feature Store capabilities such as how to create, configure a feature group, and how to ingest data into an online or offline feature store.
<a href="#">Fraud Detection with Feature Store</a>	An advanced example on how to train a fraud detection model by ingesting data into a Feature Store, querying it to form a training dataset, and how to train a simple model for inference.
<a href="#">Encrypt Data in your Online or Offline Feature Store using KMS key</a>	An advanced example on how to encrypt and decrypt data in an Online or Offline Feature Store using KMS key and how to verify that your data is encrypted. Note that this notebook tackles encryption at rest.

Notebook Title	Description
<a href="#">Client-side Encryption with Feature Store using Amazon Encryption SDK</a>	An advanced example how to do client-side encryption with Feature Store using the <a href="#">Amazon Encryption SDK library</a> which encrypts your data prior to ingesting it into your Online or Offline Feature Store.

# Train Models

For an overview on training models with Amazon SageMaker, see [Train a Model with Amazon SageMaker \(p. 8\)](#).

SageMaker provides features to monitor and manage the training and validation of machine learning models. For guidance on metrics available, incremental training, automatic model tuning, and the use of augmented manifest files to label training data, see the following topics.

- For guidance on choosing a machine learning algorithm and its implementation for your task or problem, see [Choose an Algorithm \(p. 713\)](#).
- For guidance on debugging the training of machine learning models, see [Amazon SageMaker Debugger \(p. 1579\)](#).
- For guidance on metrics used to monitor and train models, see [Monitor and Analyze Training Jobs Using Metrics \(p. 1868\)](#).
- For guidance on metrics used to detect model post-processing bias, see [Detect Posttraining Data and Model Bias \(p. 1821\)](#).
- For guidance on model explainability, see [Model Explainability \(p. 1852\)](#).
- For guidance on incremental training in SageMaker, see [Incremental Training in Amazon SageMaker \(p. 1855\)](#).
- For guidance on using managed spot training in SageMaker, see [Managed Spot Training in Amazon SageMaker \(p. 1859\)](#).
- For guidance on using training checkpoints in SageMaker, see [Use Checkpoints in Amazon SageMaker \(p. 1860\)](#).
- For guidance on automatic model tuning, also known as hyperparameter tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).
- For guidance on using an augmented manifest file to label training data, see [Provide Dataset Metadata to Training Jobs with an Augmented Manifest File \(p. 1864\)](#).

## Topics

- [Choose an Algorithm \(p. 713\)](#)
- [Manage Machine Learning with Amazon SageMaker Experiments \(p. 1553\)](#)
- [Amazon SageMaker Debugger \(p. 1579\)](#)
- [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#)
- [Distributed Training \(p. 1771\)](#)
- [Detect Posttraining Data and Model Bias \(p. 1821\)](#)
- [Model Explainability \(p. 1852\)](#)
- [Incremental Training in Amazon SageMaker \(p. 1855\)](#)
- [Managed Spot Training in Amazon SageMaker \(p. 1859\)](#)
- [Use Checkpoints in Amazon SageMaker \(p. 1860\)](#)
- [Provide Dataset Metadata to Training Jobs with an Augmented Manifest File \(p. 1864\)](#)
- [Monitor and Analyze Training Jobs Using Metrics \(p. 1868\)](#)

## Choose an Algorithm

Machine learning can help you accomplish empirical tasks that require some sort of inductive inference. This task involves induction as it uses data to train algorithms to make generalizable inferences. This

means that the algorithms can make statistically reliable predictions or decisions, or complete other tasks when applied to new data that was not used to train them.

To help you select the best algorithm for your task, we classify these tasks on various levels of abstraction. At the highest level of abstraction, machine learning attempts to find patterns or relationships between features or less structured items, such as text in a data set. Pattern recognition techniques can be classified into distinct machine learning paradigms, each of which address specific problem types. There are currently three basic paradigms for machine learning used to address various problem types:

- [Supervised learning \(p. 716\)](#)
- [Unsupervised learning \(p. 717\)](#)
- [Reinforcement learning \(p. 717\)](#)

The types of problems that each learning paradigm can address are identified by considering the inferences (or predictions, decisions, or other tasks) you want to make from the type of data that you have or could collect. Machine learning paradigms use algorithmic methods to address their various problem types. The algorithms provide recipes for solving these problems.

However, many algorithms, such as neural networks, can be deployed with different learning paradigms and on different types of problems. Multiple algorithms can also address a specific problem type. Some algorithms are more generally applicable and others are quite specific for certain kinds of objectives and data. So the mapping between machine learning algorithms and problem types is many-to-many. Also, there are various implementation options available for algorithms.

The following sections provide guidance concerning implementation options, machine learning paradigms, and algorithms appropriate for different problem types.

#### Topics

- [Choose an algorithm implementation \(p. 714\)](#)
- [Problem types for the basic machine learning paradigms \(p. 716\)](#)
- [Use Amazon SageMaker Built-in Algorithms \(p. 718\)](#)
- [Use Reinforcement Learning with Amazon SageMaker \(p. 1552\)](#)

## Choose an algorithm implementation

After choosing an algorithm, you must decide which implementation of it you want to use. Amazon SageMaker supports three implementation options that require increasing levels of effort.

- **Built-in algorithms** require the least effort and scale if the data set is large and significant resources are needed to train and deploy the model.
- If there is no built-in solution that works, try to develop one that uses **pre-made images for machine and deep learning frameworks** for supported frameworks such as Scikit-Learn, TensorFlow, PyTorch, MXNet, or Chainer.
- If you need to run custom packages or use any code which isn't a part of a supported framework or available via PyPi, then you need to build **your own custom Docker image** that is configured to install the necessary packages or software. The custom image must also be pushed to an online repository like the Amazon Elastic Container Registry.

#### Topics

- [Use a built-in algorithm \(p. 715\)](#)
- [Use script mode in a supported framework \(p. 715\)](#)
- [Use a custom Docker image \(p. 716\)](#)

## Algorithm implementation guidance

Implementation	Requires code	Pre-coded algorithms	Support for third party packages	Support for custom code	Level of effort
Built-in	No	Yes	No	No	Low
Scikit-learn	Yes	Yes	PyPi only	Yes	Medium
Spark ML	Yes	Yes	PyPi only	Yes	Medium
XGBoost (open source)	Yes	Yes	PyPi only	Yes	Medium
TensorFlow	Yes	No	PyPi only	Yes	Medium-high
PyTorch	Yes	No	PyPi only	Yes	Medium-high
MXNet	Yes	No	PyPi only	Yes	Medium-high
Chainer	Yes	No	PyPi only	Yes	Medium-high
Custom image	Yes	No	Yes, from any source	Yes	High

## Use a built-in algorithm

When choosing an algorithm for your type of problem and data, the easiest option is to use one of Amazon SageMaker's built-in algorithms. These built-in algorithms come with two major benefits.

- The built-in algorithms require no coding to start running experiments. The only inputs you need to provide are the data, hyperparameters, and compute resources. This allows you to run experiments more quickly, with less overhead for tracking results and code changes.
- The built-in algorithms come with parallelization across multiple compute instances and GPU support right out of the box for all applicable algorithms (some algorithms may not be included due to inherent limitations). If you have a lot of data with which to train your model, most built-in algorithms can easily scale to meet the demand. Even if you already have a pre-trained model, it may still be easier to use its corollary in SageMaker and input the hyper-parameters you already know than to port it over, using script mode on a supported framework.

For more information on the built-in algorithms provided by SageMaker, see [Use Amazon SageMaker Built-in Algorithms \(p. 718\)](#).

For important information about docker registry paths, data formats, recommended EC2 instance types, and CloudWatch logs common to all of the built-in algorithms provided by SageMaker, see [Common Information About Built-in Algorithms \(p. 723\)](#).

## Use script mode in a supported framework

If the algorithm you want to use for your model is not supported by a built-in choice and you are comfortable coding your own solution, then you should consider using an Amazon SageMaker supported framework. This is referred to as "script mode" because you write your custom code (script) in a text file with a .py extension. As the table above indicates, SageMaker supports most of the popular machine learning frameworks. These frameworks come preloaded with the corresponding framework and some additional Python packages, such as Pandas and NumPy, so you can write your own code for training an algorithm. These frameworks also allow you to install any Python package hosted on PyPi by including a

requirements.txt file with your training code or to include your own code directories. R is also supported natively in SageMaker notebook kernels. Some frameworks, like scikit-learn and Spark ML, have pre-coded algorithms you can use easily, while other frameworks like TensorFlow and PyTorch may require you to implement the algorithm yourself. The only limitation when using a supported framework image is that you cannot import any software packages that are not hosted on PyPi or that are not already included with the framework's image.

For more information on the frameworks supported by SageMaker, see [Use Machine Learning Frameworks, Python, and R with Amazon SageMaker \(p. 16\)](#).

## Use a custom Docker image

Amazon SageMaker's built-in algorithms and supported frameworks should cover most use cases, but there are times when you may need to use an algorithm from a package not included in any of the supported frameworks. You might also have a pre-trained model picked or persisted somewhere which you need to deploy. SageMaker uses Docker images to host the training and serving of all models, so you can supply your own custom Docker image if the package or software you need is not included in a supported framework. This may be your own Python package or an algorithm coded in a language like Stan or Julia. For these images you must also configure the training of the algorithm and serving of the model properly in your Dockerfile. This requires intermediate knowledge of Docker and is not recommended unless you are comfortable writing your own machine learning algorithm. Your Docker image must be uploaded to an online repository, such as the Amazon Elastic Container Registry (ECR) before you can train and serve your model properly.

For more information on custom Docker images in SageMaker, see [Using Docker containers with SageMaker \(p. 2134\)](#).

## Problem types for the basic machine learning paradigms

The following three sections describe the main problem types addressed by the three basic paradigms for machine learning. For a list of the built-in algorithms that SageMaker provides to address these problem types, see [Use Amazon SageMaker Built-in Algorithms \(p. 718\)](#).

### Topics

- [Supervised learning \(p. 716\)](#)
- [Unsupervised learning \(p. 717\)](#)
- [Reinforcement learning \(p. 717\)](#)

## Supervised learning

If your data set consists of features or attributes (inputs) that contain target values (outputs), then you have a supervised learning problem. If your target values are categorical (mathematically discrete), then you have a **classification problem**. It is a standard practice to distinguish binary from multiclass classification.

- **Binary classification** is a type of supervised learning that assigns an individual to one of two predefined and mutually exclusive classes based on the individual's attributes. It is supervised because the models are trained using examples in which the attributes are provided with correctly labeled objects. A medical diagnosis for whether an individual has a disease or not based on the results of diagnostic tests is an example of binary classification.
- **Multiclass classification** is a type of supervised learning that assigns an individual to one of several classes based on the individual's attributes. It is supervised because the models are trained using examples in which the attributes are provided with correctly labeled objects. An example is the

prediction of the topic most relevant to a text document. A document may be classified as being about religion, politics, or finance, or as about one of several other predefined topic classes.

If the target values you are trying to predict are mathematically continuous, then you have a **regression** problem. Regression estimates the values of a dependent target variable based on one or more other variables or attributes that are correlated with it. An example is the prediction of house prices using features like the number of bathrooms and bedrooms and the square footage of the house and garden. Regression analysis can create a model that takes one or more of these features as an input and predicts the price of a house.

For more information on the built-in supervised learning algorithms provided by SageMaker, see [Supervised Learning \(p. 721\)](#).

## Unsupervised learning

If your data set consists of features or attributes (inputs) that do not contain labels or target values (outputs), then you have an unsupervised learning problem. In this type of problem, the output must be predicted based on the pattern discovered in the input data. The goal in unsupervised learning problems is to discover patterns such groupings within the data. There are a large variety of tasks or problem types to which unsupervised learning can be applied. Principal component and cluster analyses are two of the main methods commonly deployed for preprocessing data. Here is a short list of problem types that can be addressed by unsupervised learning:

- **Dimension reduction** is typically part of a data exploration step used to determine the most relevant features to use for model construction. The idea is to transform data from a high-dimensional, sparsely populated space into a low-dimensional space that retains most significant properties of the original data. This provides relief for the curse of dimensionality that can arise with sparsely populated, high-dimensional data on which statistical analysis becomes problematic. It can also be used to help understand data, reducing high-dimensional data to a lower dimensionality that can be visualized.
- **Cluster analysis** is a class of techniques that are used to classify objects or cases into groups called clusters. It attempts to find discrete groupings within data, where members of a group are as similar as possible to one another and as different as possible from members of other groups. You define the features or attributes that you want the algorithm to use to determine similarity, select a distance function to measure similarity, and specify the number of clusters to use in the analysis.
- **Anomaly detection** is the identification of rare items, events, or observations in a data set which raise suspicions because they differ significantly from the rest of the data. The identification of anomalous items can be used, for example, to detect bank fraud or medical errors. Anomalies are also referred to as outliers, novelties, noise, deviations, and exceptions.
- **Density estimation** is the construction of estimates of unobservable underlying probability density functions based on observed data. A natural use of density estimates is for data exploration. Density estimates can discover features such as skewness and multimodality in the data. The most basic form of density estimation is a rescaled histogram.

SageMaker provides several built-in machine learning algorithms that you can use for these unsupervised learning tasks. For more information on the built-in unsupervised algorithms provided by SageMaker, see [Unsupervised Learning \(p. 721\)](#).

## Reinforcement learning

Reinforcement learning is a type of learning that is based on interaction with the environment. This type of learning is used by an agent that must learn behavior through trial-and-error interactions with a dynamic environment in which the goal is to maximize the long-term rewards that the agent receives as a result of its actions. Rewards are maximized by trading off exploring actions that have uncertain rewards with exploiting actions that have known rewards.

For more information on SageMaker's frameworks, toolkits, and environments for reinforcement learning, see [Use Reinforcement Learning with Amazon SageMaker \(p. 1552\)](#).

## Use Amazon SageMaker Built-in Algorithms

Amazon SageMaker provides a suite of built-in algorithms to help data scientists and machine learning practitioners get started on training and deploying machine learning models quickly. For someone that is new to SageMaker, choosing the right algorithm for your particular use case can be a challenging task. The following table provides a quick cheat sheet that shows how you can start with an example problem or use case and find an appropriate built-in algorithm offered by SageMaker that is valid for that problem type. Additional guidance organized by learning paradigms (supervised and unsupervised) and important data domains (text and images) is provided in the sections following the table.

**Table: Mapping use cases to built-in algorithms**

Example problems and use cases	Learning paradigm or domain	Problem types	Data input format	Built-in algorithms
Predict if an item belongs to a category: an email spam filter	Supervised Learning (p. 721)	Binary/multi-class classification	Tabular	Factorization Machines Algorithm (p. 1389), K-Nearest Neighbors (k-NN) Algorithm (p. 1428), Linear Learner Algorithm (p. 1442), XGBoost Algorithm (p. 1524)
Predict a numeric/continuous value: estimate the value of a house		Regression	Tabular	Factorization Machines Algorithm (p. 1389), K-Nearest Neighbors (k-NN) Algorithm (p. 1428), Linear Learner Algorithm (p. 1442), XGBoost Algorithm (p. 1524)
Based on historical data for a behavior, predict future behavior: predict sales on a new product based on previous sales data.		Time-series forecasting	Tabular	DeepAR Forecasting Algorithm (p. 1374)
Drop those columns from a dataset that have a weak relation with the label/target variable: the color of a car	Unsupervised Learning (p. 721)	Feature engineering: dimensionality reduction	Tabular	Principal Component Analysis (PCA) Algorithm (p. 1490)

Example problems and use cases	Learning paradigm or domain	Problem types	Data input format	Built-in algorithms
when predicting its mileage.				
Detect abnormal behavior in application: spot when an IoT sensor is sending abnormal readings		Anomaly detection	Tabular	<a href="#">Random Cut Forest (RCF) Algorithm (p. 1494)</a>
Protect your application from suspicious users: detect if an IP address accessing a service might be from a bad actor		IP anomaly detection	Tabular	<a href="#">IP Insights (p. 1410)</a>
Improve the data embeddings of the high-dimensional objects: identify duplicate support tickets or find the correct routing based on similarity of text in the tickets		Embeddings: convert high-dimensional objects into low-dimensional space.	Tabular	<a href="#">Object2Vec Algorithm (p. 1463)</a>
Group similar objects/data together: find high-, medium-, and low-spending customers from their transaction histories		Clustering or grouping	Tabular	<a href="#">K-Means Algorithm (p. 1420)</a>
Organize a set of documents into topics (not known in advance): tag a document as belonging to a medical category based on the terms used in the document.		Topic modeling	Text	<a href="#">Latent Dirichlet Allocation (LDA) Algorithm (p. 1436), Neural Topic Model (NTM) Algorithm (p. 1457)</a>

Example problems and use cases	Learning paradigm or domain	Problem types	Data input format	Built-in algorithms
Assign pre-defined categories to documents in a corpus: categorize books in a library into academic disciplines	<a href="#">Textual Analysis (p. 722)</a>	Text classification	Text	<a href="#">BlazingText algorithm (p. 1364)</a>
Convert text from one language to other: Spanish to English		Machine translation algorithm	Text	<a href="#">Sequence-to-Sequence Algorithm (p. 1512)</a>
Summarize a long text corpus: an abstract for a research paper		Text summarization	Text	<a href="#">Sequence-to-Sequence Algorithm (p. 1512)</a>
Convert audio files to text: transcribe call center conversations for further analysis		Speech-to-text	Text	<a href="#">Sequence-to-Sequence Algorithm (p. 1512)</a>
Label/tag an image based on the content of the image: alerts about adult content in an image	<a href="#">Image Processing (p. 722)</a>	Image and multi-label classification	Image	<a href="#">Image Classification Algorithm (p. 1399)</a>
Detect people and objects in an image: police review a large photo gallery for a missing person		Object detection and classification	Image	<a href="#">Object Detection Algorithm (p. 1479)</a>
Tag every pixel of an image individually with a category: self-driving cars prepare to identify objects in their way		Computer vision	Image	<a href="#">Semantic Segmentation Algorithm (p. 1502)</a>

For important information about Docker registry paths, data formats, recommended Amazon EC2 instance types, and CloudWatch logs common to all of the built-in algorithms provided by SageMaker, see [Common Information About Built-in Algorithms \(p. 723\)](#).

The following sections provide additional guidance for the Amazon SageMaker built-in algorithms grouped by the supervised and unsupervised learning paradigms to which they belong. For descriptions of these learning paradigms and their associated problem types, see [Choose an Algorithm \(p. 713\)](#). Sections are also provided for the SageMaker built-in algorithms available to address two important machine learning domains: textual analysis and image processing.

- [Supervised Learning \(p. 721\)](#)
- [Unsupervised Learning \(p. 721\)](#)
- [Textual Analysis \(p. 722\)](#)
- [Image Processing \(p. 722\)](#)

## Supervised Learning

Amazon SageMaker provides several built-in general purpose algorithms that can be used for either classification or regression problems.

- [Linear Learner Algorithm \(p. 1442\)](#)—learns a linear function for regression or a linear threshold function for classification.
- [Factorization Machines Algorithm \(p. 1389\)](#)—an extension of a linear model that is designed to economically capture interactions between features within high-dimensional sparse datasets.
- [XGBoost Algorithm \(p. 1524\)](#)—implementation of the gradient-boosted trees algorithm that combines an ensemble of estimates from a set of simpler and weaker models.
- [K-Nearest Neighbors \(k-NN\) Algorithm \(p. 1428\)](#)—a non-parametric method that uses the k nearest labeled points to assign a label to a new data point for classification or a predicted target value from the average of the k nearest points for regression.

Amazon SageMaker also provides several built-in supervised learning algorithms that are used for more specialized tasks during feature engineering and forecasting from time series data.

- [Object2Vec Algorithm \(p. 1463\)](#)—a new highly customizable multi-purpose algorithm used for feature engineering. It can learn low-dimensional dense embeddings of high-dimensional objects to produce features that improve training efficiencies for downstream models. While this is a supervised algorithm, as it requires labeled data for training, there are many scenarios in which the relationship labels can be obtained purely from natural clusterings in data, without any explicit human annotation.
- [DeepAR Forecasting Algorithm \(p. 1374\)](#)—a supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNN).

## Unsupervised Learning

Amazon SageMaker provides several built-in algorithms that can be used for a variety of unsupervised learning tasks such as clustering, dimension reduction, pattern recognition, and anomaly detection.

- [Principal Component Analysis \(PCA\) Algorithm \(p. 1490\)](#)—reduces the dimensionality (number of features) within a dataset by projecting data points onto the first few principal components. The objective is to retain as much information or variation as possible. For mathematicians, principal components are eigenvectors of the data's covariance matrix.
- [K-Means Algorithm \(p. 1420\)](#)—finds discrete groupings within data, where members of a group are as similar as possible to one another and as different as possible from members of other groups.
- [IP Insights \(p. 1410\)](#)—learns the usage patterns for IPv4 addresses. It is designed to capture associations between IPv4 addresses and various entities, such as user IDs or account numbers.
- [Random Cut Forest \(RCF\) Algorithm \(p. 1494\)](#)—detects anomalous data points within a data set that diverge from otherwise well-structured or patterned data.

## Textual Analysis

SageMaker provides algorithms that are tailored to the analysis of textual documents used in natural language processing, document classification or summarization, topic modeling or classification, and language transcription or translation.

- [BlazingText algorithm \(p. 1364\)](#)—a highly optimized implementation of the Word2vec and text classification algorithms that scale to large datasets easily. It is useful for many downstream natural language processing (NLP) tasks.
- [Sequence-to-Sequence Algorithm \(p. 1512\)](#)—a supervised algorithm commonly used for neural machine translation.
- [Latent Dirichlet Allocation \(LDA\) Algorithm \(p. 1436\)](#)—an algorithm suitable for determining topics in a set of documents. It is an *unsupervised algorithm*, which means that it doesn't use example data with answers during training.
- [Neural Topic Model \(NTM\) Algorithm \(p. 1457\)](#)—another unsupervised technique for determining topics in a set of documents, using a neural network approach.

## Image Processing

SageMaker also provides image processing algorithms that are used for image classification, object detection, and computer vision.

- [Image Classification Algorithm \(p. 1399\)](#)—uses example data with answers (referred to as a *supervised algorithm*). Use this algorithm to classify images.
- [Semantic Segmentation Algorithm \(p. 1502\)](#)—provides a fine-grained, pixel-level approach to developing computer vision applications.
- [Object Detection Algorithm \(p. 1479\)](#)—detects and classifies objects in images using a single deep neural network. It is a supervised learning algorithm that takes images as input and identifies all instances of objects within the image scene.

### Topics

- [Common Information About Built-in Algorithms \(p. 723\)](#)
- [BlazingText algorithm \(p. 1364\)](#)
- [DeepAR Forecasting Algorithm \(p. 1374\)](#)
- [Factorization Machines Algorithm \(p. 1389\)](#)
- [Image Classification Algorithm \(p. 1399\)](#)
- [IP Insights \(p. 1410\)](#)
- [K-Means Algorithm \(p. 1420\)](#)
- [K-Nearest Neighbors \(k-NN\) Algorithm \(p. 1428\)](#)
- [Latent Dirichlet Allocation \(LDA\) Algorithm \(p. 1436\)](#)
- [Linear Learner Algorithm \(p. 1442\)](#)
- [Neural Topic Model \(NTM\) Algorithm \(p. 1457\)](#)
- [Object2Vec Algorithm \(p. 1463\)](#)
- [Object Detection Algorithm \(p. 1479\)](#)
- [Principal Component Analysis \(PCA\) Algorithm \(p. 1490\)](#)
- [Random Cut Forest \(RCF\) Algorithm \(p. 1494\)](#)
- [Semantic Segmentation Algorithm \(p. 1502\)](#)
- [Sequence-to-Sequence Algorithm \(p. 1512\)](#)

- [XGBoost Algorithm \(p. 1524\)](#)

## Common Information About Built-in Algorithms

The following table lists parameters for each of the algorithms provided by Amazon SageMaker.

Algorithm name	Channel name	Training input mode	File type	Instance class	Parallelizable	
BlazingText	train	File or Pipe	Text file (one sentence per line with space-separated tokens)	GPU (single instance only) or CPU	No	
DeepAR Forecasting	train and (optionally) test	File	JSON Lines or Parquet	GPU or CPU	Yes	
Factorization Machines	train and (optionally) test	File or Pipe	recordIO-protobuf	CPU (GPU for dense data)	Yes	
Image Classification	train and validation, (optionally) train_lst, validation_lst, and model	File or Pipe	recordIO or image files (.jpg or .png)	GPU	Yes	
IP Insights	train and (optionally) validation	File	CSV	CPU or GPU	Yes	
k-means	train and (optionally) test	File or Pipe	recordIO-protobuf or CSV	CPU or GPUCommon (single GPU device on one or more instances)	No	
k-nearest-neighbor (k-NN)	train and (optionally) test	File or Pipe	recordIO-protobuf or CSV	CPU or GPU (single GPU device on one or more instances)	Yes	
LDA	train and (optionally) test	File or Pipe	recordIO-protobuf or CSV	CPU (single instance only)	No	
Linear Learner	train and (optionally) validation, test, or both	File or Pipe	recordIO-protobuf or CSV	CPU or GPU	Yes	

Algorithm name	Channel name	Training input mode	File type	Instance class	Parallelizable	
Neural Topic Model	train and (optionally) validation, test, or both	File or Pipe	recordIO-protobuf or CSV	GPU or CPU	Yes	
Object2Vec	train and (optionally) validation, test, or both	File	JSON Lines	GPU or CPU (single instance only)	No	
Object Detection	train and validation, (optionally) train_annotation, validation_annotation, and model	File or Pipe	recordIO or image files (.jpg or .png)	GPU	Yes	
PCA	train and (optionally) test	File or Pipe	recordIO-protobuf or CSV	GPU or CPU	Yes	
Random Cut Forest	train and (optionally) test	File or Pipe	recordIO-protobuf or CSV	CPU	Yes	
Semantic Segmentation	train and validation, train_annotation, validation_annotation, and (optionally) label_map and model	File or Pipe	Image files	GPU (single instance only)	No	
Seq2Seq Modeling	train, validation, and vocab	File	recordIO-protobuf	GPU (single instance only)	No	
XGBoost (0.90-1, 0.90-2, 1.0-1, 1.2-1, 1.2-21)	train and (optionally) validation	File or Pipe	CSV, LibSVM, or Parquet	CPU (or GPU for 1.2-1)	Yes	

Algorithms that are *parallelizable* can be deployed on multiple compute instances for distributed training.

The following topics provide information about Docker registry paths, data formats, recommended Amazon EC2 instance types, and CloudWatch logs common to all of the built-in algorithms provided by Amazon SageMaker.

#### Topics

- [Docker Registry Paths and Example Code \(p. 725\)](#)
- [Common Data Formats for Built-in Algorithms \(p. 1354\)](#)

- [Instance Types for Built-in Algorithms \(p. 1363\)](#)
- [Logs for Built-in Algorithms \(p. 1363\)](#)

## Docker Registry Paths and Example Code

The following topics list the Docker registry path and other parameters for each of the Amazon SageMaker provided algorithms and Deep Learning Containers (DLC).

Use the path as follows:

- To create a training job ([create\\_training\\_job](#)), specify the Docker registry path (`TrainingImage`) and the training input mode (`TrainingInputMode`) for the training image. You create a training job to train a model using a specific dataset.
- To create a model ([create\\_model](#)), specify the Docker registry path (`Image`) for the inference image (`PrimaryContainer Image`). SageMaker launches machine learning compute instances that are based on the endpoint configuration and deploys the model, which includes the artifacts (the result of model training).

### Note

For the registry path, use the `:1` version tag to ensure that you are using a stable version of the algorithm/DLC. You can reliably host a model trained using an image with the `:1` tag on an inference image that has the `:1` tag. Using the `:latest` tag in the registry path provides you with the most up-to-date version of the algorithm/DLC, but might cause problems with backward compatibility. Avoid using the `:latest` tag for production purposes.

For XGBoost, do not use `:latest` or `:1`. Use the specific version you require, such as `:0.90-1-cpu-py3`, `:0.90-2-cpu-py3`, `:1.0-1-cpu-py3`, or `:1.2-1`.

To find the registry path, choose the Amazon Region, then choose the algorithm or DLC.

### Topics

- [Docker Registry Paths and Example Code for US East \(Ohio\) \(us-east-2\) \(p. 726\)](#)
- [Docker Registry Paths and Example Code for US East \(N. Virginia\) \(us-east-1\) \(p. 753\)](#)
- [Docker Registry Paths and Example Code for US West \(N. California\) \(us-west-1\) \(p. 783\)](#)
- [Docker Registry Paths and Example Code for US West \(Oregon\) \(us-west-2\) \(p. 811\)](#)
- [Docker Registry Paths and Example Code for Africa \(Cape Town\) \(af-south-1\) \(p. 838\)](#)
- [Docker Registry Paths and Example Code for Asia Pacific \(Hong Kong\) \(ap-east-1\) \(p. 864\)](#)
- [Docker Registry Paths and Example Code for Asia Pacific \(Mumbai\) \(ap-south-1\) \(p. 891\)](#)
- [Docker Registry Paths and Example Code for Asia Pacific \(Seoul\) \(ap-northeast-2\) \(p. 919\)](#)
- [Docker Registry Paths and Example Code for Asia Pacific \(Singapore\) \(ap-southeast-1\) \(p. 946\)](#)
- [Docker Registry Paths and Example Code for Asia Pacific \(Sydney\) \(ap-southeast-2\) \(p. 974\)](#)
- [Docker Registry Paths and Example Code for Asia Pacific \(Tokyo\) \(ap-northeast-1\) \(p. 1002\)](#)
- [Docker Registry Paths and Example Code for Canada \(Central\) \(ca-central-1\) \(p. 1029\)](#)
- [Docker Registry Paths and Example Code for China \(Beijing\) \(cn-north-1\) \(p. 1057\)](#)
- [Docker Registry Paths and Example Code for China \(Ningxia\) \(cn-northwest-1\) \(p. 1083\)](#)
- [Docker Registry Paths and Example Code for Europe \(Frankfurt\) \(eu-central-1\) \(p. 1110\)](#)
- [Docker Registry Paths and Example Code for Europe \(Ireland\) \(eu-west-1\) \(p. 1138\)](#)
- [Docker Registry Paths and Example Code for Europe \(London\) \(eu-west-2\) \(p. 1165\)](#)
- [Docker Registry Paths and Example Code for Europe \(Paris\) \(eu-west-3\) \(p. 1193\)](#)
- [Docker Registry Paths and Example Code for Europe \(Stockholm\) \(eu-north-1\) \(p. 1219\)](#)
- [Docker Registry Paths and Example Code for Europe \(Milan\) \(eu-south-1\) \(p. 1246\)](#)
- [Docker Registry Paths and Example Code for Middle East \(Bahrain\) \(me-south-1\) \(p. 1272\)](#)

- Docker Registry Paths and Example Code for South America (São Paulo) (sa-east-1) (p. 1299)
- Docker Registry Paths and Example Code for Amazon GovCloud (US-West) (us-gov-west-1) (p. 1325)

## Docker Registry Paths and Example Code for US East (Ohio) (us-east-2)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

### Topics

- [BlazingText \(algorithm\) \(p. 727\)](#)
- [Chainer \(DLC\) \(p. 727\)](#)
- [Clarify \(algorithm\) \(p. 727\)](#)
- [Data Wrangler \(algorithm\) \(p. 728\)](#)
- [Debugger \(algorithm\) \(p. 728\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 728\)](#)
- [Factorization Machines \(algorithm\) \(p. 729\)](#)
- [Hugging Face \(algorithm\) \(p. 729\)](#)
- [IP Insights \(algorithm\) \(p. 730\)](#)
- [Image classification \(algorithm\) \(p. 730\)](#)
- [Inferentia MXNet \(DLC\) \(p. 730\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 730\)](#)
- [K-Means \(algorithm\) \(p. 731\)](#)
- [KNN \(algorithm\) \(p. 731\)](#)
- [LDA \(algorithm\) \(p. 731\)](#)
- [Linear Learner \(algorithm\) \(p. 732\)](#)
- [MXNet \(DLC\) \(p. 732\)](#)
- [MXNet Coach \(DLC\) \(p. 734\)](#)
- [Model Monitor \(algorithm\) \(p. 735\)](#)
- [NTM \(algorithm\) \(p. 735\)](#)
- [Neo Image Classification \(algorithm\) \(p. 735\)](#)
- [Neo MXNet \(DLC\) \(p. 736\)](#)
- [Neo PyTorch \(DLC\) \(p. 736\)](#)
- [Neo Tensorflow \(DLC\) \(p. 737\)](#)
- [Neo XGBoost \(algorithm\) \(p. 737\)](#)
- [Object Detection \(algorithm\) \(p. 737\)](#)
- [Object2Vec \(algorithm\) \(p. 738\)](#)
- [PCA \(algorithm\) \(p. 738\)](#)
- [PyTorch \(DLC\) \(p. 738\)](#)
- [Random Cut Forest \(algorithm\) \(p. 740\)](#)
- [Ray PyTorch \(DLC\) \(p. 741\)](#)
- [Scikit-learn \(algorithm\) \(p. 741\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 741\)](#)
- [Seq2Seq \(algorithm\) \(p. 742\)](#)
- [Spark \(algorithm\) \(p. 742\)](#)
- [SparkML Serving \(algorithm\) \(p. 742\)](#)
- [Tensorflow \(DLC\) \(p. 743\)](#)
- [Tensorflow Coach \(DLC\) \(p. 750\)](#)

- [Tensorflow Inferentia \(DLC\) \(p. 751\)](#)
- [Tensorflow Ray \(DLC\) \(p. 751\)](#)
- [VW \(algorithm\) \(p. 752\)](#)
- [XGBoost \(algorithm\) \(p. 752\)](#)

### [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext', region='us-east-2')
```

Registry path	Version	Job types (image scope)		
825641698319.dkr.ecr.us-1 east-2.amazonaws.com/ blazingtext:<tag>		inference, training		

### [Chainer \(DLC\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer', region='us-
east-2', version='5.0.0', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.us-1 east-2.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.us-1 east-2.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.us-1 east-2.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3

### [Clarify \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify', region='us-
east-2', version='1.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
211330385671.dkr.ecr.us-1.0 east-2.amazonaws.com/ sagemaker-clarify- processing:<tag>		processing		

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler',region='us-east-2')
```

Registry path	Version	Job types (image scope)		
415577184552.dkr.ecr.us-1.x east-2.amazonaws.com/ sagemaker- data-wrangler- container:<tag>		processing		

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger',region='us-east-2')
```

Registry path	Version	Job types (image scope)		
915447279597.dkr.ecr.us-latest east-2.amazonaws.com/ sagemaker-debugger- rules:<tag>		debugger		

### DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar',region='us-east-2')
```

Registry path	Version	Job types (image scope)		
566113047672.dkr.ecr.us-1 east-2.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
forecasting-deepar:<tag>				

### Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines', region='us-east-2')
```

Registry path	Version	Job types (image scope)		
404615174143.dkr.ecr.us-1 east-2.amazonaws.com/ factorization- machines:<tag>		inference, training		

### Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='us-  
east-2', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.us-4.4.2 east-2.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.us-4.5.0 east-2.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.us-4.4.2 east-2.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		
763104351884.dkr.ecr.us-4.5.0 east-2.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		

## IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights',region='us-east-2')
```

Registry path	Version	Job types (image scope)		
404615174143.dkr.ecr.us-1 east-2.amazonaws.com/ ipinsights:<tag>		inference, training		

## Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification',region='us-east-2')
```

Registry path	Version	Job types (image scope)		
825641698319.dkr.ecr.us-1 east-2.amazonaws.com/ image- classification:<tag>		inference, training		

## Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='us-  
east-2',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
007439368137.dkr.ecr.us-1 east-2.amazonaws.com/ sagemaker-neo- mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='us-  
east-2',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
007439368137.dkr.ecr.us-1 east-2.amazonaws.com/ sagemaker-neo- pytorch:<tag>		inference	inf	py3

### K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans', region='us-east-2')
```

Registry path	Version	Job types (image scope)		
404615174143.dkr.ecr.us-1 east-2.amazonaws.com/ kmeans:<tag>		inference, training		

### KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn', region='us-east-2')
```

Registry path	Version	Job types (image scope)		
404615174143.dkr.ecr.us-1 east-2.amazonaws.com/ knn:<tag>		inference, training		

### LDA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='lda', region='us-east-2')
```

Registry path	Version	Job types (image scope)		
999911452149.dkr.ecr.us-1 east-2.amazonaws.com/ lda:<tag>		inference, training		

## Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='us-east-2')
```

Registry path	Version	Job types (image scope)		
404615174143.dkr.ecr.us-1 east-2.amazonaws.com/ linear-learner:<tag>		inference, training		

## MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='us-  
east-2', version='1.4.1', py_version='py3', image_scope='inference',  
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>dr.310</u> east-2.amazonaws.com/ sagemaker-mxnet- eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.e <u>dr.410</u> east-2.amazonaws.com/ sagemaker-mxnet- serving-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.e <u>dr.410</u> east-2.amazonaws.com/ sagemaker-mxnet- serving:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>dr.414</u> east-2.amazonaws.com/ sagemaker-mxnet- serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e <u>0.12.1</u> east-2.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.12.1</u> east-2.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr.0 <u>6</u> -east-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.0 <u>6</u> -east-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.1 <u>6</u> -east-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1 <u>6</u> -east-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.2 <u>4</u> -east-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.2 <u>4</u> -east-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.3 <u>6</u> -east-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.3 <u>6</u> -east-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.4 <u>0</u> -east-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.4 <u>4</u> -east-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr.4 <u>4</u> -east-2.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <del>5.4</del> east-2.amazonaws.com/ mxnet-inference- eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> east-2.amazonaws.com/ mxnet-inference- eia:<tag>		eia	CPU	py3
763104351884.dkr.edr. <del>4.4</del> east-2.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>6.0</del> east-2.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> east-2.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>8.0</del> east-2.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py37
763104351884.dkr.edr. <del>4.4</del> east-2.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>6.0</del> east-2.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> east-2.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>8.0</del> east-2.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py37

## MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='coach-mxnet',region='us-east-2',version='0.11',py_version='py3',image_scope='training',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.us-east-2.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.us-east-2.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor',region='us-east-2')
```

Registry path	Version	Job types (image scope)		
777275614652.dkr.ecr.us-east-2.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm',region='us-east-2')
```

Registry path	Version	Job types (image scope)		
404615174143.dkr.ecr.us-1east-2.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='image-classification-neo',region='us-east-2')
```

Registry path	Version	Job types (image scope)		
007439368137.dkr.ecr.us-east-2.amazonaws.com/ image-classification-neo:<tag>		inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet',region='us-east-2',version='1.8',py_version='py3',image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
007439368137.dkr.ecr.us-east-2.amazonaws.com/ sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch',region='us-east-2',version='1.6',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
007439368137.dkr.ecr.us-east-2.amazonaws.com/ sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
007439368137.dkr.ecr.us-east-2.amazonaws.com/ sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
007439368137.dkr.ecr.us-east-2.amazonaws.com/		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-inference-pytorch:<tag>				

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='us-east-2', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
007439368137.dkr.ecr.us-east-2.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

### Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='us-east-2')
```

Registry path	Version	Job types (image scope)		
007439368137.dkr.ecr.us-latest		inference		

### Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='us-east-2')
```

Registry path	Version	Job types (image scope)		
825641698319.dkr.ecr.us-1		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec',region='us-east-2')
```

Registry path	Version	Job types (image scope)		
404615174143.dkr.ecr.us-1 east-2.amazonaws.com/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca',region='us-east-2')
```

Registry path	Version	Job types (image scope)		
404615174143.dkr.ecr.us-1 east-2.amazonaws.com/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch',region='us-
east-2',version='1.8.0',py_version='py3',image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.40- east-2.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.40- east-2.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e0.00- east-2.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.eifr.0 <u>6</u> -east-2.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.eifr.1 <u>6</u> -east-2.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.eifr.1 <u>6</u> -east-2.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.eifr.3 <u>4</u> -east-2.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.eifr.2 <u>6</u> -east-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.eifr.3 <u>4</u> -east-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.eifr.4 <u>6</u> -east-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.eifr.5 <u>6</u> -east-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.eifr.6 <u>6</u> -east-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.eifr.7 <u>4</u> -east-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.eifr.8 <u>6</u> -east-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <del>84</del> east-2.amazonaws.com/ pytorch- inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>26</del> east-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>34</del> east-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>46</del> east-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>56</del> east-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>66</del> east-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>74</del> east-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>86</del> east-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>84</del> east-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36

### Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='us-east-2')
```

Registry path	Version	Job types (image scope)		
404615174143.dkr.ecr.us-1 east-2.amazonaws.com/ randomcutforest:<tag>		inference, training		

### Ray PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-pytorch', region='us-
east-2', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8.5- east-2.amazonaws.com/ sagemaker-rl-ray- container:ray-0.8.5- torch-<tag>	0.8.5-	training	CPU, GPU	py36

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn', region='us-
east-2', version='0.23-1', image_scope='inference')
```

Registry path	Version	Job types (image scope)		
257758044811.dkr.ecr.us-0.20.0 east-2.amazonaws.com/ sagemaker-scikit- learn:<tag>	-0.20.0	inference, training		
257758044811.dkr.ecr.us-0.23-1 east-2.amazonaws.com/ sagemaker-scikit- learn:<tag>	-0.23-1	inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='semantic-segmentation',region='us-east-2')
```

Registry path	Version	Job types (image scope)		
825641698319.dkr.ecr.us-1 east-2.amazonaws.com/ semantic- segmentation:<tag>		inference, training		

## Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='us-east-2')
```

Registry path	Version	Job types (image scope)		
825641698319.dkr.ecr.us-1 east-2.amazonaws.com/ seq2seq:<tag>		inference, training		

## Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='us-  
east-2',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
314815235551.dkr.ecr.us-2.4 east-2.amazonaws.com/ sagemaker-spark- processing:<tag>		processing		
314815235551.dkr.ecr.us-3.0 east-2.amazonaws.com/ sagemaker-spark- processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving',region='us-east-2',version='2.4')
```

Registry path	Version	Job types (image scope)		
257758044811.dkr.ecr.us-2.2 east-2.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		
257758044811.dkr.ecr.us-2.4 east-2.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		

### Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='us-
east-2', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.10:0 east-2.amazonaws.com/ sagemaker- tensorflow- eia:<tag>		eia	CPU	py2
520713654638.dkr.ecr.11:0 east-2.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.ecr.12:0 east-2.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.ecr.13:1 east-2.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr.14:0 east-2.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.ecr.15:0 east-2.amazonaws.com/		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-tensorflow-serving-eia:<tag>				
520713654638.dkr.e-dr.1 <u>5</u> :0 east-2.amazonaws.com/ sagemaker-tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.1 <u>6</u> :0 east-2.amazonaws.com/ sagemaker-tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.1 <u>8</u> :0 east-2.amazonaws.com/ sagemaker-tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.1 <u>9</u> :0 east-2.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.1 <u>9</u> :0 east-2.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.e-dr.1 <u>4</u> :0 east-2.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.1 <u>4</u> :0 east-2.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.e-dr.1 <u>5</u> :0 east-2.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.1 <u>5</u> :0 east-2.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.e-dr.1 <u>6</u> :0 east-2.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <del>6.0</del> east-2.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> east-2.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> east-2.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> east-2.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> east-2.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>9.0</del> east-2.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>9.0</del> east-2.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <del>14.0</del> east-2.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <del>15.0</del> east-2.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.e0r. <del>0.0</del> east-2.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.3.0</u> east-2.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>0.13.0</u> east-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.14.0</u> east-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.15.0</u> east-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.15.2</u> east-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.15.3</u> east-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.15.4</u> east-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.15.5</u> east-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.6</u> east-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.4</u> east-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.2</u> east-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <del>0.05</del> east-2.amazonaws.com/ tensorflow- inference:<tag>	0.05	inference	CPU, GPU	-
763104351884.dkr.e <del>0.04</del> east-2.amazonaws.com/ tensorflow- inference:<tag>	0.04	inference	CPU, GPU	-
763104351884.dkr.e <del>0.10</del> east-2.amazonaws.com/ tensorflow- inference:<tag>	0.10	inference	CPU, GPU	-
763104351884.dkr.e <del>0.14</del> east-2.amazonaws.com/ tensorflow- inference:<tag>	0.14	inference	CPU, GPU	-
763104351884.dkr.e <del>0.12</del> east-2.amazonaws.com/ tensorflow- inference:<tag>	0.12	inference	CPU, GPU	-
763104351884.dkr.e <del>0.15</del> east-2.amazonaws.com/ tensorflow- inference:<tag>	0.15	inference	CPU, GPU	-
763104351884.dkr.e <del>0.20</del> east-2.amazonaws.com/ tensorflow- inference:<tag>	0.20	inference	CPU, GPU	-
763104351884.dkr.e <del>0.24</del> east-2.amazonaws.com/ tensorflow- inference:<tag>	0.24	inference	CPU, GPU	-
763104351884.dkr.e <del>0.22</del> east-2.amazonaws.com/ tensorflow- inference:<tag>	0.22	inference	CPU, GPU	-
763104351884.dkr.e <del>0.30</del> east-2.amazonaws.com/ tensorflow- inference:<tag>	0.30	inference	CPU, GPU	-
763104351884.dkr.e <del>0.34</del> east-2.amazonaws.com/ tensorflow- inference:<tag>	0.34	inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>2.3.2</u> east-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.4.1</u> east-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.15.1</u> east-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>2.14.0</u> east-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>2.15.0</u> east-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>2.15.2</u> east-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.e <u>2.15.3</u> east-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.e <u>2.15.4</u> east-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>2.15.5</u> east-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>2.0.0</u> east-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>2.0.1</u> east-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.02</u> -east-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.03</u> -east-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.04</u> -east-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.10</u> -east-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.14</u> -east-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.12</u> -east-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.15</u> -east-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.20</u> -east-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.24</u> -east-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.22</u> -east-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.30</u> -east-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e0.34-east-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.34-east-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.44-east-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow', region='us-east-2', version='1.0.0', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.06-east-2.amazonaws.com/sagemaker-rl-coach-container:coach-1.0.0-tf-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10-east-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10.1-east-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11-east-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.0-east-2.amazonaws.com/		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-rl-tensorflow:coach0.11.0-<tag>				
520713654638.dkr.e0.11.1-east-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inference (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow',region='us-east-2',version='1.15.0',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
007439368137.dkr.e0.15.0-east-2.amazonaws.com/sagemaker-neo-tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow',region='us-east-2',version='0.8.5',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8.2-east-2.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.2-tf-<tag>		training	CPU, GPU	py36
462105765813.dkr.e0.8.5-east-2.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-tf-<tag>		training	CPU, GPU	py36
520713654638.dkr.e0.5-is-east-2.amazonaws.com/		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-rl-tensorflow:ray0.5-<tag>				
520713654638.dkr.e0:5.3-east-2.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0:6.5-east-2.amazonaws.com/sagemaker-rl-tensorflow:ray0.6-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0:6.5-east-2.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5-<tag>		training	CPU, GPU	py3

## VW (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='vw', region='us-east-2', version='8.7.0', image_scope='training')
```

Registry path	Version	Job types (image scope)		
462105765813.dkr.ecr.us-8.7.0-east-2.amazonaws.com/sagemaker-rl-vw-container:vw-8.7.0-<tag>		training		

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost', region='us-east-2', version='1.2-1')
```

Registry path	Version	Job types (image scope)		
257758044811.dkr.ecr.us-0.90-1-east-2.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
sagemaker-xgboost:<tag>				
257758044811.dkr.ecr.us-0.90-2 east-2.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
257758044811.dkr.ecr.us-1.0-1 east-2.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
257758044811.dkr.ecr.us-1.2-1 east-2.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
257758044811.dkr.ecr.us-1.2-2 east-2.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
257758044811.dkr.ecr.us-1.3-1 east-2.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
825641698319.dkr.ecr.us-1 east-2.amazonaws.com/ xgboost:<tag>		inference, training		

## Docker Registry Paths and Example Code for US East (N. Virginia) (us-east-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

### Topics

- [BlazingText \(algorithm\) \(p. 754\)](#)
- [Chainer \(DLC\) \(p. 755\)](#)
- [Clarify \(algorithm\) \(p. 755\)](#)
- [Data Wrangler \(algorithm\) \(p. 755\)](#)
- [Debugger \(algorithm\) \(p. 756\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 756\)](#)
- [Factorization Machines \(algorithm\) \(p. 756\)](#)
- [Hugging Face \(algorithm\) \(p. 757\)](#)
- [IP Insights \(algorithm\) \(p. 757\)](#)
- [Image classification \(algorithm\) \(p. 758\)](#)
- [Inferentia MXNet \(DLC\) \(p. 758\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 758\)](#)
- [K-Means \(algorithm\) \(p. 759\)](#)

- [KNN \(algorithm\) \(p. 759\)](#)
- [LDA \(algorithm\) \(p. 759\)](#)
- [Linear Learner \(algorithm\) \(p. 760\)](#)
- [MXNet \(DLC\) \(p. 760\)](#)
- [MXNet Coach \(DLC\) \(p. 763\)](#)
- [Model Monitor \(algorithm\) \(p. 763\)](#)
- [NTM \(algorithm\) \(p. 764\)](#)
- [Neo Image Classification \(algorithm\) \(p. 764\)](#)
- [Neo MXNet \(DLC\) \(p. 764\)](#)
- [Neo PyTorch \(DLC\) \(p. 765\)](#)
- [Neo Tensorflow \(DLC\) \(p. 765\)](#)
- [Neo XGBoost \(algorithm\) \(p. 766\)](#)
- [Object Detection \(algorithm\) \(p. 766\)](#)
- [Object2Vec \(algorithm\) \(p. 766\)](#)
- [PCA \(algorithm\) \(p. 767\)](#)
- [PyTorch \(DLC\) \(p. 767\)](#)
- [Random Cut Forest \(algorithm\) \(p. 769\)](#)
- [Ray PyTorch \(DLC\) \(p. 770\)](#)
- [Scikit-learn \(algorithm\) \(p. 770\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 771\)](#)
- [Seq2Seq \(algorithm\) \(p. 771\)](#)
- [Spark \(algorithm\) \(p. 771\)](#)
- [SparkML Serving \(algorithm\) \(p. 772\)](#)
- [Tensorflow \(DLC\) \(p. 772\)](#)
- [Tensorflow Coach \(DLC\) \(p. 779\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 780\)](#)
- [Tensorflow Ray \(DLC\) \(p. 781\)](#)
- [VW \(algorithm\) \(p. 782\)](#)
- [XGBoost \(algorithm\) \(p. 782\)](#)

### [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext',region='us-east-1')

# Output path
'811284229777.dkr.ecr.us-east-1.amazonaws.com/blazingtext:1'
```

Registry path	Version	Job types (image scope)		
811284229777.dkr.ecr.us-1 east-1.amazonaws.com/ blazingtext:<tag>		inference, training		

## Chainer (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer',region='us-east-1',version='5.0.0',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')

# Output path
'520713654638.dkr.ecr.us-east-1.amazonaws.com/sagemaker-chainer:5.0.0-cpu-py3'
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.us-east-1.amazonaws.com/sagemaker-chainer:<tag>	4.0.0	inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.us-east-1.amazonaws.com/sagemaker-chainer:<tag>	4.1.0	inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.us-east-1.amazonaws.com/sagemaker-chainer:<tag>	5.0.0	inference, training	CPU, GPU	py2, py3

## Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify',region='us-east-1',version='1.0',image_scope='processing')

# Output path
'205585389593.dkr.ecr.us-east-1.amazonaws.com/sagemaker-clarify-processing:1.0'
```

Registry path	Version	Job types (image scope)		
205585389593.dkr.ecr.us-east-1.amazonaws.com/sagemaker-clarify-processing:<tag>	1.0	processing		

## Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler',region='us-east-1')
```

```
# Output path
'663277389841.dkr.ecr.us-east-1.amazonaws.com/sagemaker-data-wrangler-container:1.x'
```

Registry path	Version	Job types (image scope)		
663277389841.dkr.ecr.us-1.x east-1.amazonaws.com/ sagemaker- data-wrangler- container:<tag>		processing		

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger',region='us-east-1')

# Output path
'503895931360.dkr.ecr.us-east-1.amazonaws.com/sagemaker-debugger-rules:latest'
```

Registry path	Version	Job types (image scope)		
503895931360.dkr.ecr.us-latest east-1.amazonaws.com/ sagemaker-debugger- rules:<tag>		debugger		

### DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar',region='us-east-1')

# Output path
'522234722520.dkr.ecr.us-east-1.amazonaws.com/forecasting-deepar:1'
```

Registry path	Version	Job types (image scope)		
522234722520.dkr.ecr.us-1 east-1.amazonaws.com/ forecasting- deepar:<tag>		inference, training		

### Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines',region='us-east-1')

# Output path
'382416733822.dkr.ecr.us-east-1.amazonaws.com/factorization-machines:1'
```

Registry path	Version	Job types (image scope)		
382416733822.dkr.ecr.us-1 east-1.amazonaws.com/ factorization- machines:<tag>		inference, training		

## Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface',region='us-  
east-1',version='4.4.2',image_scope='training',base_framework_version='tensorflow2.4.1')

# Output path
'763104351884.dkr.ecr.us-east-1.amazonaws.com/huggingface-tensorflow-training:2.4.1-  
transformers4.4.2-gpu-py37'
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.us-4.4.2 east-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.us-4.5.0 east-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.us-4.4.2 east-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		
763104351884.dkr.ecr.us-4.5.0 east-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		

## IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights',region='us-east-1')

# Output path
'382416733822.dkr.ecr.us-east-1.amazonaws.com/ipinsights:1'
```

Registry path	Version	Job types (image scope)		
382416733822.dkr.ecr.us-1 east-1.amazonaws.com/ ipinsights:<tag>		inference, training		

### Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification',region='us-east-1')

# Output path
'811284229777.dkr.ecr.us-east-1.amazonaws.com/image-classification:1'
```

Registry path	Version	Job types (image scope)		
811284229777.dkr.ecr.us-1 east-1.amazonaws.com/ image- classification:<tag>		inference, training		

### Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='us-  
east-1',version='1.5.1',instance_type='ml.inf1.6xlarge')

# Output path
'785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-mxnet:1.5.1-inf-py3'
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
785573368785.dkr.ecr.us-1 east-1.amazonaws.com/ sagemaker-neo- mxnet:<tag>		inference	inf	py3

### Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch', region='us-east-1', version='1.5.1', py_version='py3')

# Output path
'785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-pytorch:1.5.1-inf-py3'
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
785573368785.dkr.ecr.us-east-1.amazonaws.com/ sagemaker-neo-pytorch:<tag>		inference	inf	py3

### K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans', region='us-east-1')

# Output path
'382416733822.dkr.ecr.us-east-1.amazonaws.com/kmeans:1'
```

Registry path	Version	Job types (image scope)		
382416733822.dkr.ecr.us-1 east-1.amazonaws.com/ kmeans:<tag>		inference, training		

### KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn', region='us-east-1')

# Output path
'382416733822.dkr.ecr.us-east-1.amazonaws.com/knn:1'
```

Registry path	Version	Job types (image scope)		
382416733822.dkr.ecr.us-1 east-1.amazonaws.com/ knn:<tag>		inference, training		

### LDA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='lda', region='us-east-1')

# Output path
'766337827248.dkr.ecr.us-east-1.amazonaws.com/lda:1'
```

Registry path	Version	Job types (image scope)		
766337827248.dkr.ecr.us-1 east-1.amazonaws.com/ lda:<tag>		inference, training		

### Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='us-east-1')

# Output path
'382416733822.dkr.ecr.us-east-1.amazonaws.com/linear-learner:1'
```

Registry path	Version	Job types (image scope)		
382416733822.dkr.ecr.us-1 east-1.amazonaws.com/ linear-learner:<tag>		inference, training		

### MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='us-  
east-1', version='1.4.1', py_version='py3', image_scope='inference',  
instance_type='ml.c5.4xlarge')

# Output path
'763104351884.dkr.ecr.us-east-1.amazonaws.com/mxnet-inference:1.4.1-cpu-py3'
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.us-1 east-1.amazonaws.com/ sagemaker-mxnet- eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.ecr.us-1 east-1.amazonaws.com/ sagemaker-mxnet- serving-eia:<tag>		eia	CPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>10</u> -east-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>11</u> -east-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e <u>0</u> . <u>12</u> :1-east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0</u> . <u>12</u> :1-east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>0</u> . <u>0</u> :1-east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0</u> . <u>0</u> :1-east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>0</u> . <u>0</u> :1-east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0</u> . <u>0</u> :1-east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>0</u> . <u>0</u> :1-east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0</u> . <u>0</u> :1-east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>0</u> . <u>0</u> :1-east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0</u> . <u>0</u> :1-east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>0</u> . <u>0</u> :1-east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <del>3.0</del> east-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <del>4.0</del> east-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <del>4.1</del> east-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <del>4.1</del> east-1.amazonaws.com/ mxnet-inference- eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <del>5.0</del> east-1.amazonaws.com/ mxnet-inference- eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> east-1.amazonaws.com/ mxnet-inference- eia:<tag>		eia	CPU	py3
763104351884.dkr.edr. <del>4.1</del> east-1.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>6.0</del> east-1.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> east-1.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>8.0</del> east-1.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py37
763104351884.dkr.edr. <del>4.1</del> east-1.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.us-east-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.ecr.us-east-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.ecr.us-east-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

### MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='us-east-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')

# Output path
'520713654638.dkr.ecr.us-east-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-cpu-py3'
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.us-east-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.ecr.us-east-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor', region='us-east-1')

# Output path
'156813124566.dkr.ecr.us-east-1.amazonaws.com/sagemaker-model-monitor-analyzer'
```

Registry path	Version	Job types (image scope)		
156813124566.dkr.ecr.us-east-1.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm', region='us-east-1')

# Output path
'382416733822.dkr.ecr.us-east-1.amazonaws.com/ntm:1'
```

Registry path	Version	Job types (image scope)		
382416733822.dkr.ecr.us-1east-1.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo', region='us-east-1')

# Output path
'785573368785.dkr.ecr.us-east-1.amazonaws.com/image-classification-neo:latest'
```

Registry path	Version	Job types (image scope)		
785573368785.dkr.ecr.us-latesteast-1.amazonaws.com/image-classification-neo:<tag>		inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet', region='us-east-1', version='1.8', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

```
# Output path
'785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-inference-mxnet:1.8-cpu-py3'
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

## Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='us-east-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')

# Output path
'785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-inference-pytorch:1.6-cpu-py3'
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

## Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='us-east-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')

# Output path
'785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-inference-tensorflow:1.15.3-cpu-py3'
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
785573368785.dkr.ecr.us-1.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

### Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='us-east-1')

# Output path
'785573368785.dkr.ecr.us-east-1.amazonaws.com/xgboost-neo:latest'
```

Registry path	Version	Job types (image scope)		
785573368785.dkr.ecr.us-latest		inference		
east-1.amazonaws.com/xgboost-neo:<tag>				

### Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='us-east-1')

# Output path
'811284229777.dkr.ecr.us-east-1.amazonaws.com/object-detection:1'
```

Registry path	Version	Job types (image scope)		
811284229777.dkr.ecr.us-1		inference, training		
east-1.amazonaws.com/object-detection:<tag>				

### Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec', region='us-east-1')

# Output path
'382416733822.dkr.ecr.us-east-1.amazonaws.com/object2vec:1'
```

Registry path	Version	Job types (image scope)		
382416733822.dkr.ecr.us-1 east-1.amazonaws.com/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca', region='us-east-1')

# Output path
'382416733822.dkr.ecr.us-east-1.amazonaws.com/pca:1'
```

Registry path	Version	Job types (image scope)		
382416733822.dkr.ecr.us-1 east-1.amazonaws.com/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch', region='us-
east-1', version='1.8.0', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')

# Output path
'763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-inference:1.8.0-cpu-py3'
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.40- east-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.40- east-1.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e0.01- east-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.eifr.0 <u>6</u> -east-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.eifr.1 <u>6</u> -east-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.eifr.1 <u>6</u> -east-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.eifr.3 <u>4</u> -east-1.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.eifr.2 <u>6</u> -east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.eifr.3 <u>4</u> -east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.eifr.4 <u>6</u> -east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.eifr.5 <u>6</u> -east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.eifr.6 <u>6</u> -east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.eifr.7 <u>4</u> -east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.eifr.8 <u>6</u> -east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-inference:<tag>	edr.84-	inference	CPU, GPU	py3, py36
763104351884.dkr.ecr.246-	east-1.amazonaws.com/pytorch-training:<tag>	training	CPU, GPU	py2, py3
763104351884.dkr.ecr.345-	east-1.amazonaws.com/pytorch-training:<tag>	training	CPU, GPU	py2, py3
763104351884.dkr.ecr.446-	east-1.amazonaws.com/pytorch-training:<tag>	training	CPU, GPU	py2, py3
763104351884.dkr.ecr.546-	east-1.amazonaws.com/pytorch-training:<tag>	training	CPU, GPU	py3
763104351884.dkr.ecr.646-	east-1.amazonaws.com/pytorch-training:<tag>	training	CPU, GPU	py3, py36
763104351884.dkr.ecr.745-	east-1.amazonaws.com/pytorch-training:<tag>	training	CPU, GPU	py3, py36
763104351884.dkr.ecr.846-	east-1.amazonaws.com/pytorch-training:<tag>	training	CPU, GPU	py3, py36
763104351884.dkr.ecr.845-	east-1.amazonaws.com/pytorch-training:<tag>	training	CPU, GPU	py3, py36

### Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='us-east-1')

# Output path
'382416733822.dkr.ecr.us-east-1.amazonaws.com/randomcutforest:1'
```

Registry path	Version	Job types (image scope)		
382416733822.dkr.ecr.us-1 east-1.amazonaws.com/ randomcutforest:<tag>		inference, training		

## Ray PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-pytorch', region='us-
east-1', version='0.8.5', instance_type='ml.c5.4xlarge')

# Output path
'462105765813.dkr.ecr.us-east-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-torch-
cpu-py36'
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8.5 east-1.amazonaws.com/ sagemaker-rl-ray- container:ray-0.8.5- torch-<tag>		training	CPU, GPU	py36

## Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn', region='us-
east-1', version='0.23-1', image_scope='inference')

# Output path
'683313688378.dkr.ecr.us-east-1.amazonaws.com/sagemaker-scikit-learn:0.23-1-cpu-py3'
```

Registry path	Version	Job types (image scope)		
683313688378.dkr.ecr.us-0.20.0 east-1.amazonaws.com/ sagemaker-scikit- learn:<tag>		inference, training		
683313688378.dkr.ecr.us-0.23-1 east-1.amazonaws.com/ sagemaker-scikit- learn:<tag>		inference, training		

## Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='us-east-1')

# Output path
'811284229777.dkr.ecr.us-east-1.amazonaws.com/semantic-segmentation:1'
```

Registry path	Version	Job types (image scope)		
811284229777.dkr.ecr.us-1 east-1.amazonaws.com/ semantic- segmentation:<tag>		inference, training		

## Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='us-east-1')

# Output path
'811284229777.dkr.ecr.us-east-1.amazonaws.com/seq2seq:1'
```

Registry path	Version	Job types (image scope)		
811284229777.dkr.ecr.us-1 east-1.amazonaws.com/ seq2seq:<tag>		inference, training		

## Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='us-  
east-1',version='3.0',image_scope='processing')

# Output path
'173754725891.dkr.ecr.us-east-1.amazonaws.com/sagemaker-spark-processing:3.0-cpu'
```

Registry path	Version	Job types (image scope)		
173754725891.dkr.ecr.us-2.4 east-1.amazonaws.com/ sagemaker-spark- processing:<tag>		processing		

Registry path	Version	Job types (image scope)		
173754725891.dkr.ecr.us-3.0 east-1.amazonaws.com/ sagemaker-spark- processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving', region='us-east-1', version='2.4')

# Output path
'683313688378.dkr.ecr.us-east-1.amazonaws.com/sagemaker-sparkml-serving:2.4'
```

Registry path	Version	Job types (image scope)		
683313688378.dkr.ecr.us-2.2 east-1.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		
683313688378.dkr.ecr.us-2.4 east-1.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='us-  
east-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')

# Output path
'520713654638.dkr.ecr.us-east-1.amazonaws.com/sagemaker-tensorflow-serving:1.12.0-cpu'
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.us-1.12.0 east-1.amazonaws.com/ sagemaker- tensorflow- eia:<tag>		eia	CPU	py2
520713654638.dkr.ecr.us-1.12.0 east-1.amazonaws.com/ sagemaker-		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow-scriptmode:<tag>				
520713654638.dkr.ecr.12.0 east-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.ecr.13.1 east-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr.14.0 east-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.ecr.15.0 east-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.ecr.16.0 east-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.ecr.17.0 east-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.ecr.18.0 east-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.ecr.19.0 east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr.19.0 east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <del>4.5</del> east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>4.5</del> east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>5.0</del> east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>5.0</del> east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>6.0</del> east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>6.0</del> east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>9.0</del> east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>10</u> .0 east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.e <u>14</u> .0 east-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>15</u> .0 east-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>0</u> .0 <u>6</u> east-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>0</u> .3 <u>6</u> east-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>1</u> .0 <u>6</u> .0 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>14</u> .0 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>15</u> .0 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>15</u> .2 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>15</u> .3 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0</u> .1 <u>5</u> .4 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .1 <u>5</u> .5 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .0 <u>6</u> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .0 <u>5</u> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .0 <u>2</u> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .0 <u>3</u> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .0 <u>4</u> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .1 <u>0</u> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .1 <u>1</u> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .1 <u>2</u> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .1 <u>3</u> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.20</u> -east-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.21</u> -east-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.22</u> -east-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.30</u> -east-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.31</u> -east-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.32</u> -east-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.41</u> -east-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.13</u> -east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.14</u> -east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.15</u> -east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.15</u> -east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.15.3</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.e <u>0.15.4</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0.15.5</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0.06</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.04</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.02</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.03</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.04</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.16</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.14</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.12</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training:<tag>	e0.15	training	CPU, GPU	py3
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training:<tag>	e0.20	training	CPU, GPU	py37
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training:<tag>	e0.24	training	CPU, GPU	py37
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training:<tag>	e0.28	training	CPU, GPU	py37
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training:<tag>	e0.30	training	CPU, GPU	py37
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training:<tag>	e0.34	training	CPU, GPU	py37
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training:<tag>	e0.38	training	CPU, GPU	py37
763104351884.dkr.ecr.us-east-1.amazonaws.com/tensorflow-training:<tag>	e0.44	training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow', region='us-east-1', version='1.0.0', image_scope='training', instance_type='ml.c5.4xlarge')

# Output path
'462105765813.dkr.ecr.us-east-1.amazonaws.com/sagemaker-rl-coach-container:coach-1.0.0-tf-cpu-py3'
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.ecr.us-east-1.amazonaws.com/sagemaker-rl-coach-container:coach-1.0.0-tf-<tag>		training	CPU, GPU	py3
520713654638.dkr.ecr.us-east-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
520713654638.dkr.ecr.us-east-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.ecr.us-east-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.ecr.us-east-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
520713654638.dkr.ecr.us-east-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inferentia (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow', region='us-east-1', version='1.15.0', instance_type='ml.inf1.6xlarge')

# Output path
'785573368785.dkr.ecr.us-east-1.amazonaws.com/sagemaker-neo-tensorflow:1.15.0-inf-py3'
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
785573368785.dkr.ecr.us-east-1.amazonaws.com/		inference	inf	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-neo-tensorflow:<tag>				

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow',region='us-east-1',version='0.8.5',instance_type='ml.c5.4xlarge')

# Output path
'462105765813.dkr.ecr.us-east-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-tf-cpu-py36'
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8s2-east-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.2-tf-<tag>		training	CPU, GPU	py36
462105765813.dkr.e0.8s5-east-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-tf-<tag>		training	CPU, GPU	py36
520713654638.dkr.e0.5s-east-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.5s3-east-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6s-east-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6s5-east-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5-<tag>		training	CPU, GPU	py3

## VW (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='vw', region='us-east-1', version='8.7.0', image_scope='training')

# Output path
'462105765813.dkr.ecr.us-east-1.amazonaws.com/sagemaker-rl-vw-container:vw-8.7.0-cpu'
```

Registry path	Version	Job types (image scope)		
462105765813.dkr.ecr.us-8.7.0 east-1.amazonaws.com/ sagemaker-rl-vw- container:vw-8.7.0- <i>&lt;tag&gt;</i>		training		

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost', region='us-east-1', version='1.2-1')

# Output path
'683313688378.dkr.ecr.us-east-1.amazonaws.com/sagemaker-xgboost:1.2-1'
```

Registry path	Version	Job types (image scope)		
683313688378.dkr.ecr.us-0.90-1 east-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
683313688378.dkr.ecr.us-0.90-2 east-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
683313688378.dkr.ecr.us-1.0-1 east-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
683313688378.dkr.ecr.us-1.2-1 east-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
683313688378.dkr.ecr.us-1.2-2 east-1.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
sagemaker-xgboost:<tag>				
683313688378.dkr.ecr.us-1.3-1 east-1.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
811284229777.dkr.ecr.us-1 east-1.amazonaws.com/ xgboost:<tag>		inference, training		

### Docker Registry Paths and Example Code for US West (N. California) (us-west-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

#### Topics

- [BlazingText \(algorithm\) \(p. 784\)](#)
- [Chainer \(DLC\) \(p. 784\)](#)
- [Clarify \(algorithm\) \(p. 785\)](#)
- [Data Wrangler \(algorithm\) \(p. 785\)](#)
- [Debugger \(algorithm\) \(p. 785\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 786\)](#)
- [Factorization Machines \(algorithm\) \(p. 786\)](#)
- [Hugging Face \(algorithm\) \(p. 786\)](#)
- [IP Insights \(algorithm\) \(p. 787\)](#)
- [Image classification \(algorithm\) \(p. 787\)](#)
- [Inferentia MXNet \(DLC\) \(p. 787\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 788\)](#)
- [K-Means \(algorithm\) \(p. 788\)](#)
- [KNN \(algorithm\) \(p. 788\)](#)
- [LDA \(algorithm\) \(p. 789\)](#)
- [Linear Learner \(algorithm\) \(p. 789\)](#)
- [MXNet \(DLC\) \(p. 789\)](#)
- [MXNet Coach \(DLC\) \(p. 792\)](#)
- [Model Monitor \(algorithm\) \(p. 792\)](#)
- [NTM \(algorithm\) \(p. 793\)](#)
- [Neo Image Classification \(algorithm\) \(p. 793\)](#)
- [Neo MXNet \(DLC\) \(p. 793\)](#)
- [Neo PyTorch \(DLC\) \(p. 793\)](#)
- [Neo Tensorflow \(DLC\) \(p. 794\)](#)
- [Neo XGBoost \(algorithm\) \(p. 794\)](#)
- [Object Detection \(algorithm\) \(p. 795\)](#)
- [Object2Vec \(algorithm\) \(p. 795\)](#)
- [PCA \(algorithm\) \(p. 795\)](#)
- [PyTorch \(DLC\) \(p. 796\)](#)

- [Random Cut Forest \(algorithm\) \(p. 798\)](#)
- [Ray PyTorch \(DLC\) \(p. 798\)](#)
- [Scikit-learn \(algorithm\) \(p. 798\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 799\)](#)
- [Seq2Seq \(algorithm\) \(p. 799\)](#)
- [Spark \(algorithm\) \(p. 799\)](#)
- [SparkML Serving \(algorithm\) \(p. 800\)](#)
- [Tensorflow \(DLC\) \(p. 800\)](#)
- [Tensorflow Coach \(DLC\) \(p. 807\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 808\)](#)
- [Tensorflow Ray \(DLC\) \(p. 809\)](#)
- [VW \(algorithm\) \(p. 809\)](#)
- [XGBoost \(algorithm\) \(p. 810\)](#)

### [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-1 west-1.amazonaws.com/ blazingtext:<tag>		inference, training		

### [Chainer \(DLC\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer', region='us-
west-1', version='5.0.0', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e4:0.0 west-1.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.e4:1.0 west-1.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.e5:0.0 west-1.amazonaws.com/		inference, training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-chainer:<tag>				

### Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify', region='us-west-1', version='1.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
740489534195.dkr.ecr.us-1.0 west-1.amazonaws.com/ sagemaker-clarify-processing:<tag>		processing		

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
926135532090.dkr.ecr.us-1.x west-1.amazonaws.com/ sagemaker-data-wrangler-container:<tag>		processing		

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
685455198987.dkr.ecr.us-latest west-1.amazonaws.com/ sagemaker-debugger-rules:<tag>		debugger		

## DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-1 west-1.amazonaws.com/ forecasting- deepar:<tag>		inference, training		

## Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-1 west-1.amazonaws.com/ factorization- machines:<tag>		inference, training		

## Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='us-  
west-1', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.us-4.4.2 west-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.us-4.5.0 west-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.us-4.4.2 west-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		
763104351884.dkr.ecr.us-4.5.0 west-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		

### IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-1 west-1.amazonaws.com/ ipinsights:<tag>		inference, training		

### Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-1 west-1.amazonaws.com/ image- classification:<tag>		inference, training		

### Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet', region='us-  
west-1', version='1.5.1', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
710691900526.dkr.ecr.us-west-1.amazonaws.com/sagemaker-neo-mxnet:<tag>		inference	inf	py3

### Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch', region='us-west-1', version='1.5.1', py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
710691900526.dkr.ecr.us-west-1.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

### K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-1-west-1.amazonaws.com/kmeans:<tag>		inference, training		

### KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-1-west-1.amazonaws.com/knn:<tag>		inference, training		

## LDA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='lda', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-1 west-1.amazonaws.com/ lda:<tag>		inference, training		

## Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-1 west-1.amazonaws.com/ linear-learner:<tag>		inference, training		

## MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='us-west-1', version='1.4.1', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr.3 <u>10</u> west-1.amazonaws.com/ sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.edr.4 <u>10</u> west-1.amazonaws.com/ sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.edr.4 <u>10</u> west-1.amazonaws.com/ sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr.4 <u>1</u> -west-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr.1 <u>2</u> .1-west-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1 <u>2</u> .1-west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.0 <u>6</u> -west-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.0 <u>6</u> -west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.1 <u>6</u> -west-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1 <u>6</u> -west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.2 <u>4</u> -west-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.2 <u>4</u> -west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.3 <u>0</u> -west-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.3 <u>0</u> -west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <b>4.0</b> -west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>4.1</b> -west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <b>4.1</b> -west-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <b>5.1</b> -west-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <b>7.0</b> -west-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr. <b>4.1</b> -west-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <b>6.0</b> -west-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>7.0</b> -west-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <b>8.0</b> -west-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37
763104351884.dkr.edr. <b>4.1</b> -west-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <b>6.0</b> -west-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.us-west-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.ecr.us-west-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

### MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='us-west-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.us-west-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.ecr.us-west-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
890145073186.dkr.ecr.us-west-1.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

## NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-west-1.amazonaws.com/ ntm:<tag>		inference, training		

## Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
710691900526.dkr.ecr.us-west-1.amazonaws.com/ image-classification-neo:<tag>	latest	inference		

## Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet', region='us-west-1', version='1.8', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
710691900526.dkr.ecr.us-west-1.amazonaws.com/ sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

## Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='us-west-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
710691900526.dkr.ecr.us-west-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
710691900526.dkr.ecr.us-west-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
710691900526.dkr.ecr.us-west-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

## Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='us-west-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
710691900526.dkr.ecr.us-west-1.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

## Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
710691900526.dkr.ecr.us-west-1.amazonaws.com/xgboost-neo:<tag>		inference		

### Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-west-1.amazonaws.com/object-detection:<tag>		inference, training		

### Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-west-1.amazonaws.com/object2vec:<tag>		inference, training		

### PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-west-1.amazonaws.com/pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch', region='us-
west-1', version='1.8.0', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.4 <u>0</u> -west-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.4 <u>0</u> -west-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.0 <u>0</u> -west-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.0 <u>0</u> -west-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.1 <u>0</u> -west-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1 <u>0</u> -west-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.3 <u>0</u> -west-1.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr.2 <u>0</u> -west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr.3 <u>0</u> -west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr.4 <u>0</u> -west-1.amazonaws.com/		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
pytorch-inference:<tag>				
763104351884.dkr.edr.5.0-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr.6.0-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr.7.0-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr.8.0-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr.8.1-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr.2.0-west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.3.0-west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.4.0-west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.5.0-west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr.6.0-west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.us-west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr.us-west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr.us-west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

### Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-1-west-1.amazonaws.com/randomcutforest:<tag>		inference, training		

### Ray PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-pytorch', region='us-west-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.ecr.us-west-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-torch-<tag>	0.8.5	training	CPU, GPU	py36

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn',region='us-west-1',version='0.23-1',image_scope='inference')
```

Registry path	Version	Job types (image scope)		
746614075791.dkr.ecr.us-0.20.0 west-1.amazonaws.com/ sagemaker-scikit- learn:<tag>		inference, training		
746614075791.dkr.ecr.us-0.23-1 west-1.amazonaws.com/ sagemaker-scikit- learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-1 west-1.amazonaws.com/ semantic- segmentation:<tag>		inference, training		

### Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='us-west-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-1 west-1.amazonaws.com/ seq2seq:<tag>		inference, training		

### Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='spark',region='us-west-1',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
667973535471.dkr.ecr.us-west-1.amazonaws.com/sagemaker-spark-processing:<tag>	2.4	processing		
667973535471.dkr.ecr.us-west-1.amazonaws.com/sagemaker-spark-processing:<tag>	3.0	processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving',region='us-west-1',version='2.4')
```

Registry path	Version	Job types (image scope)		
746614075791.dkr.ecr.us-west-1.amazonaws.com/sagemaker-sparkml-serving:<tag>	2.2	inference		
746614075791.dkr.ecr.us-west-1.amazonaws.com/sagemaker-sparkml-serving:<tag>	2.4	inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow',region='us-west-1',version='1.12.0',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.us-west-1.amazonaws.com/sagemaker-tensorflow-eia:<tag>	1.12.0	eia	CPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e-dr.1 <u>1</u> .0 west-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e-dr.1 <u>2</u> .0 west-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e-dr.1 <u>3</u> .1 west-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
520713654638.dkr.e-dr.1 <u>4</u> .0 west-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.1 <u>5</u> .0 west-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.1 <u>6</u> .0 west-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.1 <u>7</u> .0 west-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.1 <u>8</u> .0 west-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.1 <u>9</u> .0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <b>10</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>11</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>11</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>12</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>12</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>13</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>13</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>14</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>14</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>15</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>15</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>16</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>16</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>17</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>17</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>18</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>18</b> -0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>9</u> .0 <u>0</u> -west-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e <u>9</u> .0 <u>0</u> -west-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.e <u>9</u> .1 <u>4</u> .0-west-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>9</u> .1 <u>5</u> .0-west-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>9</u> .0 <u>1</u> .0-west-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>9</u> .3 <u>4</u> .0-west-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>9</u> .1 <u>5</u> .0-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>9</u> .1 <u>4</u> .0-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>9</u> .1 <u>5</u> .0-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>9</u> .1 <u>5</u> .2-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0</u> .1 <u>5</u> .3 west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .1 <u>5</u> .4 west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .1 <u>5</u> .5 west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .0 <u>6</u> west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .0 <u>5</u> west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .0 <u>2</u> west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .0 <u>3</u> west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .0 <u>4</u> west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .1 <u>0</u> west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .1 <u>1</u> west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0</u> .1 <u>2</u> west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.13</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.20</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.24</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.27</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.30</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.34</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.37</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.41</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.13-1</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.14-0</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.15-0</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>15</u> .2 west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.e <u>15</u> .3 west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.e <u>15</u> .4 west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>15</u> .5 west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0</u> . <u>0</u> .6 west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0</u> . <u>0</u> .4 west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0</u> . <u>0</u> .2 west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0</u> . <u>0</u> .3 west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0</u> . <u>0</u> .4 west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0</u> . <u>1</u> .0 west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0</u> . <u>1</u> .4 west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.12</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.15</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.20</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.24</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.27</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.30</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.34</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.37</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.41</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

## Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow', region='us-west-1', version='1.0.0', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0r.04- west-1.amazonaws.com/ sagemaker- rl-coach- container:coach-1.0.0- tf-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10- west-1.amazonaws.com/ sagemaker-rl- tensorflow:coach0.10- <tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10.1- west-1.amazonaws.com/ sagemaker-rl- tensorflow:coach0.10.1- <tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11- west-1.amazonaws.com/ sagemaker-rl- tensorflow:coach0.11- <tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.0- west-1.amazonaws.com/ sagemaker-rl- tensorflow:coach0.11.0- <tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.1- west-1.amazonaws.com/ sagemaker-rl- tensorflow:coach0.11.1- <tag>		training	CPU, GPU	py3

### Tensorflow Inferentia (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow', region='us-
west-1', version='1.15.0', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
710691900526.dkr.e0r.15.0- west-1.amazonaws.com/ sagemaker-neo- tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow', region='us-
west-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8.2-west-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.2-tf-<tag>		training	CPU, GPU	py36
462105765813.dkr.e0.8.5-west-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-tf-<tag>		training	CPU, GPU	py36
520713654638.dkr.e0.5.5-west-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.5.5-west-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6.5-west-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6.5-west-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5-<tag>		training	CPU, GPU	py3

## VW (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='vw', region='us-
west-1', version='8.7.0', image_scope='training')
```

Registry path	Version	Job types (image scope)		
462105765813.dkr.ecr.us-8.7.0 west-1.amazonaws.com/ sagemaker-rl-vw- container:vw-8.7.0- <i>&lt;tag&gt;</i>		training		

### XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost',region='us-west-1',version='1.2-1')
```

Registry path	Version	Job types (image scope)		
632365934929.dkr.ecr.us-1 west-1.amazonaws.com/ xgboost:<tag>		inference, training		
746614075791.dkr.ecr.us-0.90-1 west-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
746614075791.dkr.ecr.us-0.90-2 west-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
746614075791.dkr.ecr.us-1.0-1 west-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
746614075791.dkr.ecr.us-1.2-1 west-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
746614075791.dkr.ecr.us-1.2-2 west-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
746614075791.dkr.ecr.us-1.3-1 west-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		

## Docker Registry Paths and Example Code for US West (Oregon) (us-west-2)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

### Topics

- [BlazingText \(algorithm\) \(p. 812\)](#)
- [Chainer \(DLC\) \(p. 812\)](#)
- [Clarify \(algorithm\) \(p. 812\)](#)
- [Data Wrangler \(algorithm\) \(p. 813\)](#)
- [Debugger \(algorithm\) \(p. 813\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 813\)](#)
- [Factorization Machines \(algorithm\) \(p. 814\)](#)
- [Hugging Face \(algorithm\) \(p. 814\)](#)
- [IP Insights \(algorithm\) \(p. 815\)](#)
- [Image classification \(algorithm\) \(p. 815\)](#)
- [Inferentia MXNet \(DLC\) \(p. 815\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 815\)](#)
- [K-Means \(algorithm\) \(p. 816\)](#)
- [KNN \(algorithm\) \(p. 816\)](#)
- [LDA \(algorithm\) \(p. 816\)](#)
- [Linear Learner \(algorithm\) \(p. 817\)](#)
- [MXNet \(DLC\) \(p. 817\)](#)
- [MXNet Coach \(DLC\) \(p. 819\)](#)
- [Model Monitor \(algorithm\) \(p. 820\)](#)
- [NTM \(algorithm\) \(p. 820\)](#)
- [Neo Image Classification \(algorithm\) \(p. 820\)](#)
- [Neo MXNet \(DLC\) \(p. 821\)](#)
- [Neo PyTorch \(DLC\) \(p. 821\)](#)
- [Neo Tensorflow \(DLC\) \(p. 822\)](#)
- [Neo XGBoost \(algorithm\) \(p. 822\)](#)
- [Object Detection \(algorithm\) \(p. 822\)](#)
- [Object2Vec \(algorithm\) \(p. 823\)](#)
- [PCA \(algorithm\) \(p. 823\)](#)
- [PyTorch \(DLC\) \(p. 823\)](#)
- [Random Cut Forest \(algorithm\) \(p. 825\)](#)
- [Ray PyTorch \(DLC\) \(p. 826\)](#)
- [Scikit-learn \(algorithm\) \(p. 826\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 826\)](#)
- [Seq2Seq \(algorithm\) \(p. 827\)](#)
- [Spark \(algorithm\) \(p. 827\)](#)
- [SparkML Serving \(algorithm\) \(p. 827\)](#)
- [Tensorflow \(DLC\) \(p. 828\)](#)
- [Tensorflow Coach \(DLC\) \(p. 835\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 836\)](#)

- [Tensorflow Ray \(DLC\) \(p. 836\)](#)
- [VW \(algorithm\) \(p. 837\)](#)
- [XGBoost \(algorithm\) \(p. 837\)](#)

### [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext',region='us-west-2')
```

Registry path	Version	Job types (image scope)		
433757028032.dkr.ecr.us-1 west-2.amazonaws.com/ blazingtext:<tag>		inference, training		

### [Chainer \(DLC\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer',region='us-
west-2',version='5.0.0',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.us-1 west-2.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.us-1 west-2.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.us-1 west-2.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3

### [Clarify \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify',region='us-
west-2',version='1.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
306415355426.dkr.ecr.us-1.0 west-2.amazonaws.com/ sagemaker-clarify- processing:<tag>		processing		

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler',region='us-west-2')
```

Registry path	Version	Job types (image scope)		
174368400705.dkr.ecr.us-1.x west-2.amazonaws.com/ sagemaker- data-wrangler- container:<tag>		processing		

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger',region='us-west-2')
```

Registry path	Version	Job types (image scope)		
895741380848.dkr.ecr.us-latest west-2.amazonaws.com/ sagemaker-debugger- rules:<tag>		debugger		

### DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar',region='us-west-2')
```

Registry path	Version	Job types (image scope)		
156387875391.dkr.ecr.us-1 west-2.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
forecasting-deepar:<tag>				

### Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines', region='us-west-2')
```

Registry path	Version	Job types (image scope)		
174872318107.dkr.ecr.us-1 west-2.amazonaws.com/ factorization- machines:<tag>		inference, training		

### Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='us-  
west-2', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.us-4.4.2 west-2.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.us-4.5.0 west-2.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.us-4.4.2 west-2.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		
763104351884.dkr.ecr.us-4.5.0 west-2.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		

## [IP Insights \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights',region='us-west-2')
```

Registry path	Version	Job types (image scope)		
174872318107.dkr.ecr.us-1 west-2.amazonaws.com/ ipinsights:<tag>		inference, training		

## [Image classification \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification',region='us-west-2')
```

Registry path	Version	Job types (image scope)		
433757028032.dkr.ecr.us-1 west-2.amazonaws.com/ image- classification:<tag>		inference, training		

## [Inferentia MXNet \(DLC\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='us-  
west-2',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
301217895009.dkr.ecr.us-1 west-2.amazonaws.com/ sagemaker-neo- mxnet:<tag>		inference	inf	py3

## [Inferentia PyTorch \(DLC\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='us-  
west-2',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
301217895009.dkr.ecr.us-1 west-2.amazonaws.com/ sagemaker-neo- pytorch:<tag>		inference	inf	py3

### K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans', region='us-west-2')
```

Registry path	Version	Job types (image scope)		
174872318107.dkr.ecr.us-1 west-2.amazonaws.com/ kmeans:<tag>		inference, training		

### KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn', region='us-west-2')
```

Registry path	Version	Job types (image scope)		
174872318107.dkr.ecr.us-1 west-2.amazonaws.com/ knn:<tag>		inference, training		

### LDA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='lda', region='us-west-2')
```

Registry path	Version	Job types (image scope)		
266724342769.dkr.ecr.us-1 west-2.amazonaws.com/ lda:<tag>		inference, training		

## Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='us-west-2')
```

Registry path	Version	Job types (image scope)		
174872318107.dkr.ecr.us-1 west-2.amazonaws.com/ linear-learner:<tag>		inference, training		

## MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='us-west-2', version='1.4.1', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr.3 <u>10</u> west-2.amazonaws.com/ sagemaker-mxnet- eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.edr.4 <u>10</u> west-2.amazonaws.com/ sagemaker-mxnet- serving-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.edr.4 <u>10</u> west-2.amazonaws.com/ sagemaker-mxnet- serving:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.4 <u>14</u> west-2.amazonaws.com/ sagemaker-mxnet- serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e <u>0.12</u> .1 west-2.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.12</u> .1 west-2.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr.0 <u>6</u> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.0 <u>6</u> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.1 <u>6</u> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1 <u>6</u> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.2 <u>4</u> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.2 <u>4</u> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.3 <u>6</u> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.3 <u>6</u> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.4 <u>0</u> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.4 <u>4</u> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr.4 <u>4</u> -west-2.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <del>5.4</del> west-2.amazonaws.com/ mxnet-inference- eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> west-2.amazonaws.com/ mxnet-inference- eia:<tag>		eia	CPU	py3
763104351884.dkr.edr. <del>4.4</del> west-2.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>6.0</del> west-2.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> west-2.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>8.0</del> west-2.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py37
763104351884.dkr.edr. <del>4.4</del> west-2.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>6.0</del> west-2.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> west-2.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>8.0</del> west-2.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py37

## MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='coach-mxnet',region='us-west-2',version='0.11',py_version='py3',image_scope='training',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.us-west-2.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.us-west-2.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor',region='us-west-2')
```

Registry path	Version	Job types (image scope)		
159807026194.dkr.ecr.us-west-2.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm',region='us-west-2')
```

Registry path	Version	Job types (image scope)		
174872318107.dkr.ecr.us-1-west-2.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='image-classification-neo',region='us-west-2')
```

Registry path	Version	Job types (image scope)		
301217895009.dkr.ecr.us-west-2.amazonaws.com/image-classification-neo:<tag>		inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet',region='us-west-2',version='1.8',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch',region='us-west-2',version='1.6',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
301217895009.dkr.ecr.us-west-2.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
301217895009.dkr.ecr.us-west-2.amazonaws.com/		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-inference-pytorch:<tag>				

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='us-west-2', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
301217895009.dkr.ecr.us-15-3-west-2.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

### Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='us-west-2')
```

Registry path	Version	Job types (image scope)		
301217895009.dkr.ecr.us-latest-west-2.amazonaws.com/xgboost-neo:<tag>		inference		

### Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='us-west-2')
```

Registry path	Version	Job types (image scope)		
433757028032.dkr.ecr.us-1-west-2.amazonaws.com/object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec', region='us-west-2')
```

Registry path	Version	Job types (image scope)		
174872318107.dkr.ecr.us-1 west-2.amazonaws.com/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca', region='us-west-2')
```

Registry path	Version	Job types (image scope)		
174872318107.dkr.ecr.us-1 west-2.amazonaws.com/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch', region='us-
west-2', version='1.8.0', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.40- west-2.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.40- west-2.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e0.01- west-2.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.eifr.0 <u>6</u> -west-2.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.eifr.1 <u>6</u> -west-2.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.eifr.1 <u>6</u> -west-2.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.eifr.3 <u>4</u> -west-2.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.eifr.2 <u>6</u> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.eifr.3 <u>4</u> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.eifr.4 <u>6</u> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.eifr.5 <u>6</u> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.eifr.6 <u>6</u> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.eifr.7 <u>4</u> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.eifr.8 <u>6</u> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <del>84</del> west-2.amazonaws.com/ pytorch- inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>20</del> west-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>34</del> west-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>40</del> west-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>50</del> west-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>60</del> west-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>74</del> west-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>80</del> west-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>84</del> west-2.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36

### Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='us-west-2')
```

Registry path	Version	Job types (image scope)		
174872318107.dkr.ecr.us-1 west-2.amazonaws.com/ randomcutforest:<tag>		inference, training		

### Ray PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-pytorch', region='us-
west-2', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8.5- west-2.amazonaws.com/ sagemaker-rl-ray- container:ray-0.8.5- torch-<tag>	0.8.5-	training	CPU, GPU	py36

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn', region='us-
west-2', version='0.23-1', image_scope='inference')
```

Registry path	Version	Job types (image scope)		
246618743249.dkr.ecr.us-0.20.0 west-2.amazonaws.com/ sagemaker-scikit- learn:<tag>	-0.20.0	inference, training		
246618743249.dkr.ecr.us-0.23-1 west-2.amazonaws.com/ sagemaker-scikit- learn:<tag>	-0.23-1	inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='semantic-segmentation',region='us-west-2')
```

Registry path	Version	Job types (image scope)		
433757028032.dkr.ecr.us-1 west-2.amazonaws.com/ semantic- segmentation:<tag>		inference, training		

## Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='us-west-2')
```

Registry path	Version	Job types (image scope)		
433757028032.dkr.ecr.us-1 west-2.amazonaws.com/ seq2seq:<tag>		inference, training		

## Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='us-
west-2',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
153931337802.dkr.ecr.us-2.4 west-2.amazonaws.com/ sagemaker-spark- processing:<tag>		processing		
153931337802.dkr.ecr.us-3.0 west-2.amazonaws.com/ sagemaker-spark- processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving',region='us-west-2',version='2.4')
```

Registry path	Version	Job types (image scope)		
246618743249.dkr.ecr.us-west-2.amazonaws.com/sagemaker-sparkml-serving:<tag>		inference		
246618743249.dkr.ecr.us-west-2.amazonaws.com/sagemaker-sparkml-serving:<tag>		inference		

### Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='us-west-2', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.us-west-2.amazonaws.com/sagemaker-tensorflow-eia:<tag>		eia	CPU	py2
520713654638.dkr.ecr.us-west-2.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.ecr.us-west-2.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.ecr.us-west-2.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr.us-west-2.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>		eia	CPU	-
520713654638.dkr.ecr.us-west-2.amazonaws.com/		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-tensorflow-serving-eia:<tag>				
520713654638.dkr.e-dr.1 <u>5</u> :0 west-2.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.1 <u>6</u> :0 west-2.amazonaws.com/sagemaker-tensorflow-serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.1 <u>8</u> :0 west-2.amazonaws.com/sagemaker-tensorflow-serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.1 <u>9</u> :0 west-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.1 <u>0</u> :0 west-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.e-dr.1 <u>1</u> - west-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.1 <u>2</u> - west-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.e-dr.1 <u>3</u> - west-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.1 <u>4</u> - west-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <del>6.0</del> west-2.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> west-2.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> west-2.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> west-2.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> west-2.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>9.0</del> west-2.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>9.0</del> west-2.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <del>14.0</del> west-2.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <del>15.0</del> west-2.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <del>16.0</del> west-2.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.3.0</u> west-2.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>0.13.0</u> west-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.14.0</u> west-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.15.0</u> west-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.15.2</u> west-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.15.3</u> west-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.15.4</u> west-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.15.5</u> west-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.6</u> west-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.4</u> west-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.2</u> west-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.05</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.04</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.16</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.14</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.12</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.11</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.10</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.09</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.08</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.07</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.3<u>2</u></u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.4<u>5</u></u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.1<u>5</u>.1</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.1<u>4</u>.0</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.1<u>5</u>.0</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.1<u>5</u>.2</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.e <u>0.1<u>5</u>.3</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.e <u>0.1<u>5</u>.4</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0.1<u>5</u>.5</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0.0<u>6</u></u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.0<u>4</u></u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.02</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.03</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.04</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.10</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.14</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.12</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.15</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.20</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.24</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.22</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.30</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e0.34-west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.35-west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.45-west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow', region='us-west-2', version='1.0.0', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.06-west-2.amazonaws.com/sagemaker-rl-coach-container:coach-1.0.0-tf-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10-west-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10-1-west-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11-west-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11-0-west-2.amazonaws.com/		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-rl-tensorflow:coach0.11.0-<tag>				
520713654638.dkr.e0.15.1-west-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

### Tensorflow Inference (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow',region='us-west-2',version='1.15.0',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
301217895009.dkr.e0.15.0-west-2.amazonaws.com/sagemaker-neo-tensorflow:<tag>		inference	inf	py3

### Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow',region='us-west-2',version='0.8.5',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8.2-west-2.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.2-tf-<tag>		training	CPU, GPU	py36
462105765813.dkr.e0.8.5-west-2.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-tf-<tag>		training	CPU, GPU	py36
520713654638.dkr.e0.5.1-west-2.amazonaws.com/		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-rl-tensorflow:ray0.5- <i>&lt;tag&gt;</i>				
520713654638.dkr.e0:5.3-west-2.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3- <i>&lt;tag&gt;</i>		training	CPU, GPU	py3
520713654638.dkr.e0:6.5-west-2.amazonaws.com/sagemaker-rl-tensorflow:ray0.6- <i>&lt;tag&gt;</i>		training	CPU, GPU	py3
520713654638.dkr.e0:6.5-west-2.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5- <i>&lt;tag&gt;</i>		training	CPU, GPU	py3

## VW (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='vw', region='us-west-2', version='8.7.0', image_scope='training')
```

Registry path	Version	Job types (image scope)		
462105765813.dkr.ecr.us-west-2.amazonaws.com/sagemaker-rl-vw-container:vw-8.7.0- <i>&lt;tag&gt;</i>		training		

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost', region='us-west-2', version='1.2-1')
```

Registry path	Version	Job types (image scope)		
246618743249.dkr.ecr.us-west-2.amazonaws.com/	0.90-1	inference, training		

Registry path	Version	Job types (image scope)		
sagemaker-xgboost:<tag>				
246618743249.dkr.ecr.us-west-2.amazonaws.com/sagemaker-xgboost:<tag>	0.90-2	inference, training		
246618743249.dkr.ecr.us-west-2.amazonaws.com/sagemaker-xgboost:<tag>	1.0-1	inference, training		
246618743249.dkr.ecr.us-west-2.amazonaws.com/sagemaker-xgboost:<tag>	1.2-1	inference, training		
246618743249.dkr.ecr.us-west-2.amazonaws.com/sagemaker-xgboost:<tag>	1.2-2	inference, training		
246618743249.dkr.ecr.us-west-2.amazonaws.com/sagemaker-xgboost:<tag>	1.3-1	inference, training		
433757028032.dkr.ecr.us-west-2.amazonaws.com/xgboost:<tag>	1	inference, training		

## Docker Registry Paths and Example Code for Africa (Cape Town) (af-south-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

### Topics

- [BlazingText \(algorithm\) \(p. 839\)](#)
- [Chainer \(DLC\) \(p. 839\)](#)
- [Clarify \(algorithm\) \(p. 840\)](#)
- [Data Wrangler \(algorithm\) \(p. 840\)](#)
- [Debugger \(algorithm\) \(p. 841\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 841\)](#)
- [Factorization Machines \(algorithm\) \(p. 841\)](#)
- [Hugging Face \(algorithm\) \(p. 841\)](#)
- [IP Insights \(algorithm\) \(p. 842\)](#)
- [Image classification \(algorithm\) \(p. 842\)](#)
- [Inferentia MXNet \(DLC\) \(p. 843\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 843\)](#)
- [K-Means \(algorithm\) \(p. 843\)](#)

- [KNN \(algorithm\) \(p. 843\)](#)
- [Linear Learner \(algorithm\) \(p. 844\)](#)
- [MXNet \(DLC\) \(p. 844\)](#)
- [MXNet Coach \(DLC\) \(p. 847\)](#)
- [Model Monitor \(algorithm\) \(p. 847\)](#)
- [NTM \(algorithm\) \(p. 847\)](#)
- [Neo Image Classification \(algorithm\) \(p. 848\)](#)
- [Neo MXNet \(DLC\) \(p. 848\)](#)
- [Neo PyTorch \(DLC\) \(p. 848\)](#)
- [Neo Tensorflow \(DLC\) \(p. 849\)](#)
- [Neo XGBoost \(algorithm\) \(p. 849\)](#)
- [Object Detection \(algorithm\) \(p. 850\)](#)
- [Object2Vec \(algorithm\) \(p. 850\)](#)
- [PCA \(algorithm\) \(p. 850\)](#)
- [PyTorch \(DLC\) \(p. 850\)](#)
- [Random Cut Forest \(algorithm\) \(p. 853\)](#)
- [Scikit-learn \(algorithm\) \(p. 853\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 853\)](#)
- [Seq2Seq \(algorithm\) \(p. 854\)](#)
- [Spark \(algorithm\) \(p. 854\)](#)
- [SparkML Serving \(algorithm\) \(p. 854\)](#)
- [Tensorflow \(DLC\) \(p. 855\)](#)
- [Tensorflow Coach \(DLC\) \(p. 862\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 863\)](#)
- [Tensorflow Ray \(DLC\) \(p. 863\)](#)
- [XGBoost \(algorithm\) \(p. 864\)](#)

### BlazingText (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext',region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1.south-1.amazonaws.com/blazingtext:<tag>		inference, training		

### Chainer (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='chainer', region='af-south-1', version='5.0.0', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
313743910680.dkr.ecr.af-0.0 south-1.amazonaws.com/ sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3
313743910680.dkr.ecr.1.0 south-1.amazonaws.com/ sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3
313743910680.dkr.ecr.5.0.0 south-1.amazonaws.com/ sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3

### Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify', region='af-south-1', version='1.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
811711786498.dkr.ecr.af-1.0 south-1.amazonaws.com/ sagemaker-clarify-processing:<tag>		processing		

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler', region='af-south-1')
```

Registry path	Version	Job types (image scope)		
143210264188.dkr.ecr.af-1.x south-1.amazonaws.com/ sagemaker-data-wrangler-container:<tag>		processing		

## Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger', region='af-south-1')
```

Registry path	Version	Job types (image scope)		
314341159256.dkr.ecr.af-latest south-1.amazonaws.com/ sagemaker-debugger- rules:<tag>		debugger		

## DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar', region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1 south-1.amazonaws.com/ forecasting- deepar:<tag>		inference, training		

## Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines', region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1 south-1.amazonaws.com/ factorization- machines:<tag>		inference, training		

## Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='af-  
south-1', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
626614931356.dkr.ecr.af-4.4.2 south-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
626614931356.dkr.ecr.af-4.5.0 south-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
626614931356.dkr.ecr.af-4.4.2 south-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		
626614931356.dkr.ecr.af-4.5.0 south-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		

### IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights', region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1 south-1.amazonaws.com/ ipinsights:<tag>		inference, training		

### Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification', region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1 south-1.amazonaws.com/ image- classification:<tag>		inference, training		

## Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='af-south-1',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
774647643957.dkr.ecr.af-south-1.amazonaws.com/sagemaker-neo-mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='af-south-1',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
774647643957.dkr.ecr.af-south-1.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

## K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans',region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1-south-1.amazonaws.com/kmeans:<tag>		inference, training		

## KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn',region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1.south-1.amazonaws.com/knn:<tag>		inference, training		

### Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1.south-1.amazonaws.com/linear-learner:<tag>		inference, training		

### MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='af-south-1', version='1.4.1', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
313743910680.dkr.ecr.af-1.south-1.amazonaws.com/sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3
313743910680.dkr.ecr.af-1.south-1.amazonaws.com/sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
313743910680.dkr.ecr.af-1.south-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
313743910680.dkr.ecr.af-1.south-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
313743910680.dkr.e0.12.1 south-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
313743910680.dkr.e0.12.1 south-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
313743910680.dkr.edr.0.0 south-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
313743910680.dkr.edr.0.0 south-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
313743910680.dkr.edr.1.0 south-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
313743910680.dkr.edr.1.0 south-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
313743910680.dkr.edr.2.0 south-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
313743910680.dkr.edr.2.0 south-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
313743910680.dkr.edr.3.0 south-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
313743910680.dkr.edr.3.0 south-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
313743910680.dkr.edr.4.0 south-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
313743910680.dkr.ecr. <del>4.11</del> south-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2
626614931356.dkr.ecr. <del>4.11</del> south-1.amazonaws.com/ mxnet-inference- eia:<tag>		eia	CPU	py2, py3
626614931356.dkr.ecr. <del>5.11</del> south-1.amazonaws.com/ mxnet-inference- eia:<tag>		eia	CPU	py2, py3
626614931356.dkr.ecr. <del>7.10</del> south-1.amazonaws.com/ mxnet-inference- eia:<tag>		eia	CPU	py3
626614931356.dkr.ecr. <del>4.11</del> south-1.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py3
626614931356.dkr.ecr. <del>6.10</del> south-1.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py2, py3
626614931356.dkr.ecr. <del>7.10</del> south-1.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py3
626614931356.dkr.ecr. <del>8.10</del> south-1.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py37
626614931356.dkr.ecr. <del>4.11</del> south-1.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py3
626614931356.dkr.ecr. <del>6.10</del> south-1.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py2, py3
626614931356.dkr.ecr. <del>7.10</del> south-1.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
626614931356.dkr.ecr.af-south-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

### MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='af-south-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
313743910680.dkr.ecr.af-south-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
313743910680.dkr.ecr.af-south-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor', region='af-south-1')
```

Registry path	Version	Job types (image scope)		
875698925577.dkr.ecr.af-south-1.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='ntm', region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1.south-1.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo', region='af-south-1')
```

Registry path	Version	Job types (image scope)		
774647643957.dkr.ecr.af-latest.south-1.amazonaws.com/image-classification-neo:<tag>		inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet', region='af-south-1', version='1.8', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
774647643957.dkr.ecr.af-south-1.amazonaws.com/sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='af-south-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
774647643957.dkr.ecr.af-south-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
774647643957.dkr.ecr.af-south-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
774647643957.dkr.ecr.af-south-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='af-south-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
774647643957.dkr.ecr.af-south-1.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

### Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='af-south-1')
```

Registry path	Version	Job types (image scope)		
774647643957.dkr.ecr.af-latest.south-1.amazonaws.com/xgboost-neo:<tag>		inference		

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection',region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1 south-1.amazonaws.com/ object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec',region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1 south-1.amazonaws.com/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca',region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1 south-1.amazonaws.com/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch',region='af-
south-1',version='1.8.0',py_version='py3',image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
313743910680.dkr.e0.4.0 south-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
313743910680.dkr.e0.4.0 south-1.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
313743910680.dkr.edr.0.0 south-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
313743910680.dkr.edr.0.0 south-1.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
313743910680.dkr.edr.1.0 south-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
313743910680.dkr.edr.1.0 south-1.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
626614931356.dkr.edr.3.0 south-1.amazonaws.com/ pytorch-inference- eia:<tag>		eia	CPU	py3
626614931356.dkr.edr.3.0 south-1.amazonaws.com/ pytorch- inference:<tag>		inference	CPU, GPU	py2, py3
626614931356.dkr.edr.3.0 south-1.amazonaws.com/ pytorch- inference:<tag>		inference	CPU, GPU	py2, py3
626614931356.dkr.edr.4.0 south-1.amazonaws.com/ pytorch- inference:<tag>		inference	CPU, GPU	py3
626614931356.dkr.edr.5.0 south-1.amazonaws.com/ pytorch- inference:<tag>		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
626614931356.dkr.edr. <del>6.0</del> south-1.amazonaws.com/ pytorch- inference:<tag>		inference	CPU, GPU	py3, py36
626614931356.dkr.edr. <del>7.0</del> south-1.amazonaws.com/ pytorch- inference:<tag>		inference	CPU, GPU	py3, py36
626614931356.dkr.edr. <del>8.0</del> south-1.amazonaws.com/ pytorch- inference:<tag>		inference	CPU, GPU	py3, py36
626614931356.dkr.edr. <del>8.1</del> south-1.amazonaws.com/ pytorch- inference:<tag>		inference	CPU, GPU	py3, py36
626614931356.dkr.edr. <del>2.0</del> south-1.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py2, py3
626614931356.dkr.edr. <del>3.0</del> south-1.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py2, py3
626614931356.dkr.edr. <del>4.0</del> south-1.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py2, py3
626614931356.dkr.edr. <del>5.0</del> south-1.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3
626614931356.dkr.edr. <del>6.0</del> south-1.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36
626614931356.dkr.edr. <del>7.0</del> south-1.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36
626614931356.dkr.edr. <del>8.0</del> south-1.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
626614931356.dkr.ecr.af-1.south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

### Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest',region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1.south-1.amazonaws.com/randomcutforest:<tag>		inference, training		

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn',region='af-south-1',version='0.23-1',image_scope='inference')
```

Registry path	Version	Job types (image scope)		
510948584623.dkr.ecr.af-0.20.0.south-1.amazonaws.com/sagemaker-scikit-learn:<tag>		inference, training		
510948584623.dkr.ecr.af-0.23-1.south-1.amazonaws.com/sagemaker-scikit-learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1 south-1.amazonaws.com/ semantic-segmentation:<tag>		inference, training		

### Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='af-south-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1 south-1.amazonaws.com/ seq2seq:<tag>		inference, training		

### Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='af-south-1',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
309385258863.dkr.ecr.af-2.4 south-1.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		
309385258863.dkr.ecr.af-3.0 south-1.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		

### SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving',region='af-south-1',version='2.4')
```

Registry path	Version	Job types (image scope)		
510948584623.dkr.ecr.af-2.2 south-1.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		
510948584623.dkr.ecr.af-2.4 south-1.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		

### Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='af-south-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
313743910680.dkr.ecr.10-0 south-1.amazonaws.com/ sagemaker- tensorflow- eia:<tag>		eia	CPU	py2
313743910680.dkr.ecr.11-0 south-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
313743910680.dkr.ecr.12-0 south-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
313743910680.dkr.ecr.13-1 south-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
313743910680.dkr.ecr.14-0 south-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
313743910680.dkr.ecr.15-0 south-1.amazonaws.com/		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-tensorflow-serving-eia:<tag>				
313743910680.dkr.e-dr.15.0 south-1.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>		eia	CPU	-
313743910680.dkr.e-dr.16.0 south-1.amazonaws.com/sagemaker-tensorflow-serving:<tag>		inference	CPU, GPU	-
313743910680.dkr.e-dr.17.0 south-1.amazonaws.com/sagemaker-tensorflow-serving:<tag>		inference	CPU, GPU	-
313743910680.dkr.e-dr.18.0 south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
313743910680.dkr.e-dr.19.0 south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
313743910680.dkr.e-dr.19.1 south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
313743910680.dkr.e-dr.19.1 south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
313743910680.dkr.e-dr.5.0 south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
313743910680.dkr.e-dr.5.0 south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
313743910680.dkr.e-dr.6.0 south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
313743910680.dkr.edr. <del>6.0</del> south-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
313743910680.dkr.edr. <del>7.0</del> south-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
313743910680.dkr.edr. <del>7.0</del> south-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
313743910680.dkr.edr. <del>8.0</del> south-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
313743910680.dkr.edr. <del>8.0</del> south-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
313743910680.dkr.edr. <del>9.0</del> south-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
313743910680.dkr.edr. <del>9.0</del> south-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
626614931356.dkr.edr. <del>14.0</del> south-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
626614931356.dkr.edr. <del>15.0</del> south-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
626614931356.dkr.edr. <del>16.0</del> south-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
626614931356.dkr.e <u>0.3.0</u> south-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
626614931356.dkr.e <u>0.3.0</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.4.0</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.5.0</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.5.2</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.5.3</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.5.4</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.5.5</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.6.0</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.6.1</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.6.2</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
626614931356.dkr.e <u>0.03</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.04</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.10</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.11</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.12</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.13</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.14</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.15</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.16</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.17</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.18</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.19</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.20</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.21</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.22</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.23</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.24</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
626614931356.dkr.e <u>0.3.2</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.e <u>0.4.1</u> south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
626614931356.dkr.ed <u>0.15.1</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
626614931356.dkr.ed <u>0.14.0</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
626614931356.dkr.ed <u>0.15.0</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
626614931356.dkr.ed <u>0.15.2</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
626614931356.dkr.ed <u>0.15.3</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
626614931356.dkr.ed <u>0.15.4</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
626614931356.dkr.ed <u>0.15.5</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
626614931356.dkr.e <u>0.0.9</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
626614931356.dkr.e <u>0.0.1</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
626614931356.dkr.e <u>0.0.2</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
626614931356.dkr.e <u>0.0.3</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
626614931356.dkr.e <u>0.0.4</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
626614931356.dkr.e <u>0.0.10</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
626614931356.dkr.e <u>0.0.11</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
626614931356.dkr.e <u>0.0.12</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
626614931356.dkr.e <u>0.0.13</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
626614931356.dkr.e <u>0.0.19</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py37
626614931356.dkr.e <u>0.0.21</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py37
626614931356.dkr.e <u>0.0.22</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py37
626614931356.dkr.e <u>0.0.30</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
626614931356.dkr.e0.3.11-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
626614931356.dkr.e0.3.12-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
626614931356.dkr.e0.4.11-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow', region='af-south-1', version='1.0.0', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
313743910680.dkr.e0.10-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
313743910680.dkr.e0.10.1-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
313743910680.dkr.e0.11-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
313743910680.dkr.e0.11.0-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
313743910680.dkr.e0.11.1-south-1.amazonaws.com/sagemaker-rl-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow:coach0.11.1- <i>&lt;tag&gt;</i>				

## Tensorflow Inference (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow', region='af-south-1', version='1.15.0', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
774647643957.dkr.ecr.us-west-2. south-1.amazonaws.com/ sagemaker-neo-tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow', region='af-south-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
313743910680.dkr.ecr.us-west-2. south-1.amazonaws.com/ sagemaker-rl-tensorflow:ray0.5-<tag>		training	CPU, GPU	py3
313743910680.dkr.ecr.us-west-2. south-1.amazonaws.com/ sagemaker-rl-tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
313743910680.dkr.ecr.us-west-2. south-1.amazonaws.com/ sagemaker-rl-tensorflow:ray0.6-<tag>		training	CPU, GPU	py3
313743910680.dkr.ecr.us-west-2. south-1.amazonaws.com/ sagemaker-rl-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow:ray0.6.5-<tag>				

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost', region='af-south-1', version='1.2-1')
```

Registry path	Version	Job types (image scope)		
455444449433.dkr.ecr.af-1.south-1.amazonaws.com/xgboost:<tag>		inference, training		
510948584623.dkr.ecr.af-0.90-1.south-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
510948584623.dkr.ecr.af-0.90-2.south-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
510948584623.dkr.ecr.af-1.0-1.south-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
510948584623.dkr.ecr.af-1.2-1.south-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
510948584623.dkr.ecr.af-1.2-2.south-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
510948584623.dkr.ecr.af-1.3-1.south-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		

## Docker Registry Paths and Example Code for Asia Pacific (Hong Kong) (ap-east-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

### Topics

- [BlazingText \(algorithm\) \(p. 865\)](#)
- [Chainer \(DLC\) \(p. 866\)](#)
- [Clarify \(algorithm\) \(p. 866\)](#)
- [Data Wrangler \(algorithm\) \(p. 867\)](#)
- [Debugger \(algorithm\) \(p. 867\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 867\)](#)
- [Factorization Machines \(algorithm\) \(p. 867\)](#)
- [Hugging Face \(algorithm\) \(p. 868\)](#)
- [IP Insights \(algorithm\) \(p. 868\)](#)
- [Image classification \(algorithm\) \(p. 869\)](#)
- [Inferentia MXNet \(DLC\) \(p. 869\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 869\)](#)
- [K-Means \(algorithm\) \(p. 869\)](#)
- [KNN \(algorithm\) \(p. 870\)](#)
- [Linear Learner \(algorithm\) \(p. 870\)](#)
- [MXNet \(DLC\) \(p. 870\)](#)
- [MXNet Coach \(DLC\) \(p. 873\)](#)
- [Model Monitor \(algorithm\) \(p. 873\)](#)
- [NTM \(algorithm\) \(p. 874\)](#)
- [Neo Image Classification \(algorithm\) \(p. 874\)](#)
- [Neo MXNet \(DLC\) \(p. 874\)](#)
- [Neo PyTorch \(DLC\) \(p. 875\)](#)
- [Neo Tensorflow \(DLC\) \(p. 875\)](#)
- [Neo XGBoost \(algorithm\) \(p. 876\)](#)
- [Object Detection \(algorithm\) \(p. 876\)](#)
- [Object2Vec \(algorithm\) \(p. 876\)](#)
- [PCA \(algorithm\) \(p. 876\)](#)
- [PyTorch \(DLC\) \(p. 877\)](#)
- [Random Cut Forest \(algorithm\) \(p. 879\)](#)
- [Scikit-learn \(algorithm\) \(p. 879\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 880\)](#)
- [Seq2Seq \(algorithm\) \(p. 880\)](#)
- [Spark \(algorithm\) \(p. 880\)](#)
- [SparkML Serving \(algorithm\) \(p. 881\)](#)
- [Tensorflow \(DLC\) \(p. 881\)](#)
- [Tensorflow Coach \(DLC\) \(p. 888\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 889\)](#)
- [Tensorflow Ray \(DLC\) \(p. 889\)](#)
- [XGBoost \(algorithm\) \(p. 890\)](#)

## [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext', region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-east-1.amazonaws.com/blazingtext:<tag>		inference, training		

### Chainer (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer', region='ap-east-1', version='5.0.0', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
057415533634.dkr.ecr.ap-east-1.amazonaws.com/sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3
057415533634.dkr.ecr.ap-east-1.amazonaws.com/sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3
057415533634.dkr.ecr.ap-east-1.amazonaws.com/sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3

### Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify', region='ap-east-1', version='1.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
098760798382.dkr.ecr.ap-1.0-east-1.amazonaws.com/sagemaker-clarify-processing:<tag>		processing		

## Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler',region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
707077482487.dkr.ecr.ap-1.x east-1.amazonaws.com/ sagemaker- data-wrangler- container:<tag>		processing		

## Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger',region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
199566480951.dkr.ecr.ap-latest east-1.amazonaws.com/ sagemaker-debugger- rules:<tag>		debugger		

## DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar',region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-1 east-1.amazonaws.com/ forecasting- deepar:<tag>		inference, training		

## Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines',region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-1 east-1.amazonaws.com/ factorization- machines:<tag>		inference, training		

## Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='ap-
east-1', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
871362719292.dkr.ecr.ap-4.4.2 east-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
871362719292.dkr.ecr.ap-4.5.0 east-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
871362719292.dkr.ecr.ap-4.4.2 east-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		
871362719292.dkr.ecr.ap-4.5.0 east-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		

## IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights', region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-1 east-1.amazonaws.com/ ipinsights:<tag>		inference, training		

## Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification', region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-east-1.amazonaws.com/image-classification:<tag>		inference, training		

## Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet', region='ap-east-1', version='1.5.1', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
110948597952.dkr.ecr.ap-east-1.amazonaws.com/sagemaker-neo-mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch', region='ap-east-1', version='1.5.1', py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
110948597952.dkr.ecr.ap-east-1.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

## K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='kmeans', region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-1 east-1.amazonaws.com/ kmeans:<tag>		inference, training		

### KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn', region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-1 east-1.amazonaws.com/ knn:<tag>		inference, training		

### Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-1 east-1.amazonaws.com/ linear-learner:<tag>		inference, training		

### MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='ap-
east-1', version='1.4.1', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
057415533634.dkr.ecr.ap-0 east-1.amazonaws.com/		eia	CPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-mxnet-eia:<tag>				
057415533634.dkr.e <u>10</u> -east-1.amazonaws.com/sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
057415533634.dkr.e <u>10</u> -east-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
057415533634.dkr.e <u>10</u> -east-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2
057415533634.dkr.e <u>0.10.1</u> -east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
057415533634.dkr.e <u>0.10.1</u> -east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
057415533634.dkr.e <u>0.10</u> -east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
057415533634.dkr.e <u>0.10</u> -east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
057415533634.dkr.e <u>10</u> -east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
057415533634.dkr.e <u>10</u> -east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
057415533634.dkr.e <u>20</u> -east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
057415533634.dkr.ecr.us-east-1.amazonaws.com/sagemaker-mxnet:<tag>	edr.2.0-	training	CPU, GPU	py2, py3
057415533634.dkr.ecr.us-east-1.amazonaws.com/sagemaker-mxnet:<tag>	edr.3.0-	inference	CPU, GPU	py2, py3
057415533634.dkr.ecr.us-east-1.amazonaws.com/sagemaker-mxnet:<tag>	edr.3.0-	training	CPU, GPU	py2, py3
057415533634.dkr.ecr.us-east-1.amazonaws.com/sagemaker-mxnet:<tag>	edr.4.0-	training	CPU, GPU	py2, py3
057415533634.dkr.ecr.us-east-1.amazonaws.com/sagemaker-mxnet:<tag>	edr.4.0-	training	CPU, GPU	py2
871362719292.dkr.ecr.us-east-1.amazonaws.com/mxnet-inference-eia:<tag>	edr.4.0-	eia	CPU	py2, py3
871362719292.dkr.ecr.us-east-1.amazonaws.com/mxnet-inference-eia:<tag>	edr.5.0-	eia	CPU	py2, py3
871362719292.dkr.ecr.us-east-1.amazonaws.com/mxnet-inference-eia:<tag>	edr.7.0-	eia	CPU	py3
871362719292.dkr.ecr.us-east-1.amazonaws.com/mxnet-inference:<tag>	edr.4.0-	inference	CPU, GPU	py3
871362719292.dkr.ecr.us-east-1.amazonaws.com/mxnet-inference:<tag>	edr.6.0-	inference	CPU, GPU	py2, py3
871362719292.dkr.ecr.us-east-1.amazonaws.com/mxnet-inference:<tag>	edr.7.0-	inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
871362719292.dkr.e0.1p.0-east-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37
871362719292.dkr.e0.1p.0-east-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
871362719292.dkr.e0.1p.0-east-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
871362719292.dkr.e0.1p.0-east-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
871362719292.dkr.e0.1p.0-east-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

## MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='ap-east-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
057415533634.dkr.e0.1p.0-east-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
057415533634.dkr.e0.1p.0-east-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

## Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='model-monitor',region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
001633400207.dkr.ecr.ap-east-1.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm',region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-east-1.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo',region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
110948597952.dkr.ecr.ap-east-1.amazonaws.com/image-classification-neo:<tag>	latest	inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet',region='ap-east-1',version='1.8',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
110948597952.dkr.ecr.ap-east-1.amazonaws.com/sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='ap-east-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
110948597952.dkr.ecr.ap-east-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
110948597952.dkr.ecr.ap-east-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
110948597952.dkr.ecr.ap-east-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='ap-east-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
110948597952.dkr.ecr.ap.3east-1.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

## Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
110948597952.dkr.ecr.ap-east-1.amazonaws.com/xgboost-neo:<tag>		inference		

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-east-1.amazonaws.com/object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec', region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-east-1.amazonaws.com/object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca', region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-1.east-1.amazonaws.com/pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch', region='ap-east-1', version='1.8.0', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
057415533634.dkr.e0.40-east-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
057415533634.dkr.e0.40-east-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
057415533634.dkr.edr.0.0-east-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
057415533634.dkr.edr.0.0-east-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
057415533634.dkr.edr.1.0-east-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
057415533634.dkr.edr.1.0-east-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
871362719292.dkr.edr.3.0-east-1.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
871362719292.dkr.edr.2.0-east-1.amazonaws.com/		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
pytorch-inference:<tag>				
871362719292.dkr.e-dr.3.1-p-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
871362719292.dkr.e-dr.4.0-p-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
871362719292.dkr.e-dr.5.0-p-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
871362719292.dkr.e-dr.6.0-p-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
871362719292.dkr.e-dr.7.0-p-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
871362719292.dkr.e-dr.8.0-p-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
871362719292.dkr.e-dr.8.1-p-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
871362719292.dkr.e-dr.2.0-p-east-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
871362719292.dkr.e-dr.3.1-p-east-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
871362719292.dkr.e-dr.4.0-p-east-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
871362719292.dkr.ecr.ap-east-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
871362719292.dkr.ecr.ap-east-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
871362719292.dkr.ecr.ap-east-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
871362719292.dkr.ecr.ap-east-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
871362719292.dkr.ecr.ap-east-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

## Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-east-1.amazonaws.com/randomcutforest:<tag>		inference, training		

## Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn', region='ap-east-1', version='0.23-1', image_scope='inference')
```

Registry path	Version	Job types (image scope)		
651117190479.dkr.ecr.ap-0.20.0-east-1.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
sagemaker-scikit-learn:<tag>				
651117190479.dkr.ecr.ap-0.23-1 east-1.amazonaws.com/ sagemaker-scikit-learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-1 east-1.amazonaws.com/ semantic-segmentation:<tag>		inference, training		

### Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='ap-east-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-1 east-1.amazonaws.com/ seq2seq:<tag>		inference, training		

### Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='ap-east-1',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
732049463269.dkr.ecr.ap-2.4 east-1.amazonaws.com/		processing		

Registry path	Version	Job types (image scope)		
sagemaker-spark-processing:<tag>				
732049463269.dkr.ecr.ap-3.0 east-1.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving', region='ap-east-1', version='2.4')
```

Registry path	Version	Job types (image scope)		
651117190479.dkr.ecr.ap-2.2 east-1.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		
651117190479.dkr.ecr.ap-2.4 east-1.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='ap-east-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
057415533634.dkr.ecr.ap-0.0 east-1.amazonaws.com/ sagemaker-tensorflow-eia:<tag>		eia	CPU	py2
057415533634.dkr.ecr.ap-0.0 east-1.amazonaws.com/ sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
057415533634.dkr.ecr. <sup>10</sup> .0 east-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
057415533634.dkr.ecr. <sup>10</sup> .1 east-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
057415533634.dkr.ecr. <sup>10</sup> .0 east-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
057415533634.dkr.ecr. <sup>10</sup> .0 east-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
057415533634.dkr.ecr. <sup>10</sup> .0 east-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
057415533634.dkr.ecr. <sup>10</sup> .0 east-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
057415533634.dkr.ecr. <sup>10</sup> .0 east-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
057415533634.dkr.ecr. <sup>10</sup> .0 east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
057415533634.dkr.ecr. <sup>10</sup> .0 east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
057415533634.dkr.edr. <sup>4.0</sup> -east-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
057415533634.dkr.edr. <sup>4.0</sup> -east-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
057415533634.dkr.edr. <sup>5.0</sup> -east-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
057415533634.dkr.edr. <sup>5.0</sup> -east-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
057415533634.dkr.edr. <sup>6.0</sup> -east-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
057415533634.dkr.edr. <sup>6.0</sup> -east-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
057415533634.dkr.edr. <sup>7.0</sup> -east-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
057415533634.dkr.edr. <sup>7.0</sup> -east-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
057415533634.dkr.edr. <sup>8.0</sup> -east-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
057415533634.dkr.edr. <sup>8.0</sup> -east-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
057415533634.dkr.edr. <sup>9.0</sup> -east-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
057415533634.dkr.ecr.us-east-1.amazonaws.com/sagemaker-tensorflow:<tag>	edr.9.0-	training	CPU, GPU	py2
871362719292.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-eia:<tag>	edr.14.0	eia	CPU	-
871362719292.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-eia:<tag>	edr.15.0	eia	CPU	-
871362719292.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-eia:<tag>	edr.16.0	eia	CPU	-
871362719292.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference-eia:<tag>	edr.17.0	eia	CPU	-
871362719292.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference:<tag>	edr.15.0	inference	CPU, GPU	-
871362719292.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference:<tag>	edr.16.0	inference	CPU, GPU	-
871362719292.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference:<tag>	edr.17.0	inference	CPU, GPU	-
871362719292.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference:<tag>	edr.15.2	inference	CPU, GPU	-
871362719292.dkr.ecr.us-east-1.amazonaws.com/tensorflow-inference:<tag>	edr.15.3	inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
871362719292.dkr.e <u>0.15</u> .4 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <u>0.15</u> .5 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <u>0.16</u> - east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <u>0.16</u> - east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <u>0.16</u> - east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <u>0.16</u> - east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <u>0.16</u> - east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <u>0.16</u> - east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <u>0.16</u> - east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <u>0.16</u> - east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <u>0.16</u> - east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
871362719292.dkr.e <del>0.20</del> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <del>0.21</del> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <del>0.22</del> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <del>0.30</del> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <del>0.31</del> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <del>0.32</del> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <del>0.41</del> east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
871362719292.dkr.e <del>0.10.1</del> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
871362719292.dkr.e <del>0.10.0</del> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
871362719292.dkr.e <del>0.15.0</del> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
871362719292.dkr.e <del>0.15.2</del> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
871362719292.dkr.e <u>0.15.3</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
871362719292.dkr.e <u>0.15.4</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
871362719292.dkr.e <u>0.15.5</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
871362719292.dkr.e <u>0.16.0</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
871362719292.dkr.e <u>0.16.1</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
871362719292.dkr.e <u>0.16.2</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
871362719292.dkr.e <u>0.16.3</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
871362719292.dkr.e <u>0.16.4</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
871362719292.dkr.e <u>0.16.5</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
871362719292.dkr.e <u>0.17.0</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
871362719292.dkr.e0.13-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
871362719292.dkr.e0.20-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
871362719292.dkr.e0.21-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
871362719292.dkr.e0.22-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
871362719292.dkr.e0.30-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
871362719292.dkr.e0.31-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
871362719292.dkr.e0.32-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
871362719292.dkr.e0.41-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow',region='ap-east-1',version='1.0.0',image_scope='training',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
057415533634.dkr.e0.10-east-1.amazonaws.com/sagemaker-rl-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow:coach0.10-<tag>				
057415533634.dkr.e0.10.1 east-1.amazonaws.com/ sagemaker-rl- tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
057415533634.dkr.e0.11p- east-1.amazonaws.com/ sagemaker-rl- tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3
057415533634.dkr.e0.11p.0 east-1.amazonaws.com/ sagemaker-rl- tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
057415533634.dkr.e0.11p.1 east-1.amazonaws.com/ sagemaker-rl- tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inference (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow', region='ap-east-1', version='1.15.0', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
110948597952.dkr.e0.15.0 east-1.amazonaws.com/ sagemaker-neo- tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow', region='ap-east-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
057415533634.dkr.e0.5p-east-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5-<tag>		training	CPU, GPU	py3
057415533634.dkr.e0.5p3-east-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
057415533634.dkr.e0.6p-east-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6-<tag>		training	CPU, GPU	py3
057415533634.dkr.e0.6p5-east-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5-<tag>		training	CPU, GPU	py3

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost',region='ap-east-1',version='1.2-1')
```

Registry path	Version	Job types (image scope)		
286214385809.dkr.ecr.ap-1east-1.amazonaws.com/xgboost:<tag>		inference, training		
651117190479.dkr.ecr.ap-0.90-1east-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
651117190479.dkr.ecr.ap-0.90-2east-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
651117190479.dkr.ecr.ap-1.0-1east-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		

Registry path	Version	Job types (image scope)		
651117190479.dkr.ecr.ap-1.2-1 east-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
651117190479.dkr.ecr.ap-1.2-2 east-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
651117190479.dkr.ecr.ap-1.3-1 east-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		

### Docker Registry Paths and Example Code for Asia Pacific (Mumbai) (ap-south-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

#### Topics

- [BlazingText \(algorithm\) \(p. 892\)](#)
- [Chainer \(DLC\) \(p. 892\)](#)
- [Clarify \(algorithm\) \(p. 893\)](#)
- [Data Wrangler \(algorithm\) \(p. 893\)](#)
- [Debugger \(algorithm\) \(p. 893\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 894\)](#)
- [Factorization Machines \(algorithm\) \(p. 894\)](#)
- [Hugging Face \(algorithm\) \(p. 894\)](#)
- [IP Insights \(algorithm\) \(p. 895\)](#)
- [Image classification \(algorithm\) \(p. 895\)](#)
- [Inferentia MXNet \(DLC\) \(p. 896\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 896\)](#)
- [K-Means \(algorithm\) \(p. 896\)](#)
- [KNN \(algorithm\) \(p. 896\)](#)
- [LDA \(algorithm\) \(p. 897\)](#)
- [Linear Learner \(algorithm\) \(p. 897\)](#)
- [MXNet \(DLC\) \(p. 897\)](#)
- [MXNet Coach \(DLC\) \(p. 900\)](#)
- [Model Monitor \(algorithm\) \(p. 900\)](#)
- [NTM \(algorithm\) \(p. 901\)](#)
- [Neo Image Classification \(algorithm\) \(p. 901\)](#)
- [Neo MXNet \(DLC\) \(p. 901\)](#)
- [Neo PyTorch \(DLC\) \(p. 902\)](#)
- [Neo Tensorflow \(DLC\) \(p. 902\)](#)
- [Neo XGBoost \(algorithm\) \(p. 903\)](#)

- [Object Detection \(algorithm\) \(p. 903\)](#)
- [Object2Vec \(algorithm\) \(p. 903\)](#)
- [PCA \(algorithm\) \(p. 903\)](#)
- [PyTorch \(DLC\) \(p. 904\)](#)
- [Random Cut Forest \(algorithm\) \(p. 906\)](#)
- [Ray PyTorch \(DLC\) \(p. 906\)](#)
- [Scikit-learn \(algorithm\) \(p. 907\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 907\)](#)
- [Seq2Seq \(algorithm\) \(p. 907\)](#)
- [Spark \(algorithm\) \(p. 908\)](#)
- [SparkML Serving \(algorithm\) \(p. 908\)](#)
- [Tensorflow \(DLC\) \(p. 908\)](#)
- [Tensorflow Coach \(DLC\) \(p. 916\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 916\)](#)
- [Tensorflow Ray \(DLC\) \(p. 917\)](#)
- [VW \(algorithm\) \(p. 918\)](#)
- [XGBoost \(algorithm\) \(p. 918\)](#)

### [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-south-1.amazonaws.com/blazingtext:<tag>		inference, training		

### [Chainer \(DLC\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer', region='ap-south-1', version='5.0.0', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-chainer:<tag>	v1.0.0-	inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-chainer:<tag>	v5.0.0-	inference, training	CPU, GPU	py2, py3

### Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify',region='ap-south-1',version='1.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
452307495513.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-clarify-processing:<tag>	v1.0.0-	processing		

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler',region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
089933028263.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-data-wrangler-container:<tag>	v1.x	processing		

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger',region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
904829902805.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-debugger-rules:<tag>		debugger		

### DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-south-1.amazonaws.com/forecasting-deepar:<tag>		inference, training		

### Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-south-1.amazonaws.com/factorization-machines:<tag>		inference, training		

### Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='ap-south-1', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.ap-south-1.amazonaws.com/	4.4.2	training		

Registry path	Version	Job types (image scope)		
huggingface-pytorch-training:<tag>				
763104351884.dkr.ecr.ap-4.5.0 south-1.amazonaws.com/huggingface-pytorch-training:<tag>		training		
763104351884.dkr.ecr.ap-4.4.2 south-1.amazonaws.com/huggingface-tensorflow-training:<tag>		training		
763104351884.dkr.ecr.ap-4.5.0 south-1.amazonaws.com/huggingface-tensorflow-training:<tag>		training		

## IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-1 south-1.amazonaws.com/ipinsights:<tag>		inference, training		

## Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-1 south-1.amazonaws.com/image-classification:<tag>		inference, training		

## Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='ap-south-1',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
763008648453.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-neo-mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='ap-south-1',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
763008648453.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

## K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans',region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-south-1.amazonaws.com/kmeans:<tag>		inference, training		

## KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn',region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-1 south-1.amazonaws.com/ knn:<tag>		inference, training		

## LDA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='lda', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-1 south-1.amazonaws.com/ lda:<tag>		inference, training		

## Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-1 south-1.amazonaws.com/ linear-learner:<tag>		inference, training		

## MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='ap-south-1', version='1.4.1', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-1 south-1.amazonaws.com/ sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>dr.10-</u> south-1.amazonaws.com/ sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.e <u>dr.10-</u> south-1.amazonaws.com/ sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>dr.10-</u> south-1.amazonaws.com/ sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e <u>dr.10.-1</u> south-1.amazonaws.com/ sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>dr.10.-1</u> south-1.amazonaws.com/ sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>dr.10-</u> south-1.amazonaws.com/ sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>dr.10-</u> south-1.amazonaws.com/ sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>dr.10-</u> south-1.amazonaws.com/ sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>dr.10-</u> south-1.amazonaws.com/ sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>dr.10-</u> south-1.amazonaws.com/ sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>dr.20-</u> south-1.amazonaws.com/ sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <sup>3.0-</sup> south-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <sup>3.0-</sup> south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <sup>4.0-</sup> south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <sup>4.0-</sup> south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <sup>4.0-</sup> south-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <sup>4.0-</sup> south-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <sup>7.0-</sup> south-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr. <sup>4.0-</sup> south-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <sup>6.0-</sup> south-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr. <sup>7.0-</sup> south-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <sup>8.0-</sup> south-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e0.ap-south-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e0.ap-south-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e0.ap-south-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e0.ap-south-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

## MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='ap-south-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.ap-south-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.ap-south-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

## Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
126357580389.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-1-south-1.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
763008648453.dkr.ecr.ap-latest-south-1.amazonaws.com/image-classification-neo:<tag>		inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet', region='ap-south-1', version='1.8', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
763008648453.dkr.ecr.ap-south-1.amazonaws.com/		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-inference-mxnet:<tag>				

## Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='ap-south-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
763008648453.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
763008648453.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
763008648453.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

## Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='ap-south-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
763008648453.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

## Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
763008648453.dkr.ecr.ap-south-1.amazonaws.com/xgboost-neo:<tag>		inference		

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-south-1.amazonaws.com/object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-south-1.amazonaws.com/object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-1.south-1.amazonaws.com/pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch', region='ap-south-1', version='1.8.0', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.40-south-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.40-south-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0-south-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0-south-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0-south-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0-south-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.3.0-south-1.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr.2.0-south-1.amazonaws.com/		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
pytorch-inference:<tag>				
763104351884.dkr.e-dr.ap-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.e-dr.ap-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.e-dr.ap-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.e-dr.ap-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.e-dr.ap-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.e-dr.ap-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.e-dr.ap-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.e-dr.ap-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e-dr.ap-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e-dr.ap-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.ap-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.ecr.ap-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr.ap-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr.ap-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr.ap-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

## Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-south-1.amazonaws.com/randomcutforest:<tag>		inference, training		

## Ray PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-pytorch', region='ap-south-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.ecr.ap-south-1.amazonaws.com/		training	CPU, GPU	py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-rl-ray-container:ray-0.8.5-torch-<tag>				

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn',region='ap-south-1',version='0.23-1',image_scope='inference')
```

Registry path	Version	Job types (image scope)		
720646828776.dkr.ecr.ap-0.20.0-south-1.amazonaws.com/sagemaker-scikit-learn:<tag>		inference, training		
720646828776.dkr.ecr.ap-0.23-1-south-1.amazonaws.com/sagemaker-scikit-learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-1-south-1.amazonaws.com/semantic-segmentation:<tag>		inference, training		

### Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='ap-south-1')
```

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-1 south-1.amazonaws.com/ seq2seq:<tag>		inference, training		

## Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark', region='ap-south-1', version='3.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
105495057255.dkr.ecr.ap-2.4 south-1.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		
105495057255.dkr.ecr.ap-3.0 south-1.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving', region='ap-south-1', version='2.4')
```

Registry path	Version	Job types (image scope)		
720646828776.dkr.ecr.ap-2.2 south-1.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		
720646828776.dkr.ecr.ap-2.4 south-1.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='tensorflow', region='ap-south-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-tensorflow-eia:<tag>	1.10.0	eia	CPU	py2
520713654638.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>	1.10.0	training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>	1.10.0	training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>	1.10.1	training	CPU, GPU	py2
520713654638.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>	1.10.0	eia	CPU	-
520713654638.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>	1.10.0	eia	CPU	-
520713654638.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>	1.10.0	eia	CPU	-
520713654638.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-tensorflow-serving:<tag>	1.10.0	inference	CPU, GPU	-
520713654638.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-	1.10.0	inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow-serving:<tag>				
520713654638.dkr.ecr. <sup>10.0</sup> south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <sup>10.0</sup> south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr. <sup>11</sup> south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <sup>11</sup> south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr. <sup>10</sup> south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <sup>10</sup> south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr. <sup>10</sup> south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <sup>10</sup> south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr. <sup>10</sup> south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <sup>10</sup> south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr. <sup>10</sup> south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <sup>10</sup> south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <sup>20</sup> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <sup>20</sup> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <sup>20</sup> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <sup>20</sup> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <sup>14</sup> .0-south-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <sup>15</sup> .0-south-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <sup>16</sup> .0-south-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <sup>17</sup> .0-south-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <sup>18</sup> .0-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.edr. <sup>19</sup> .0-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr.15.0 south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.edr.15.2 south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.edr.15.3 south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.edr.15.4 south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.edr.15.5 south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e2r.0.0- south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e2r.0.1- south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e2r.0.2- south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e2r.0.3- south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e2r.0.4- south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e2r.1.0- south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.1</u> 4-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.1</u> 2-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.1</u> 3-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.2</u> 0-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.2</u> 1-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.2</u> 2-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.3</u> 0-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.3</u> 1-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.3</u> 2-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.4</u> 1-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.1</u> 1-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <b>14</b> .0 south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>15</b> .0 south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>15</b> .2 south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.edr. <b>15</b> .3 south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.edr. <b>15</b> .4 south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.edr. <b>15</b> .5 south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.edr. <b>16</b> .0- south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>16</b> .1- south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>16</b> .2- south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>16</b> .3- south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <b>16</b> .4- south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e0.10-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e0.11-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e0.12-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e0.13-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e0.20-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.21-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.22-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.30-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.31-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.32-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.41-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

## Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow',region='ap-south-1',version='1.0.0',image_scope='training',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.0.0-ap-south-1.amazonaws.com/sagemaker-rl-coach-container:coach-1.0.0-tf-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10-ap-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10.1-ap-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11-ap-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.0-ap-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.1-ap-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inferentia (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow',region='ap-south-1',version='1.15.0',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
763008648453.dkr.e0.1p.0 south-1.amazonaws.com/ sagemaker-neo- tensorflow:<tag>		inference	inf	py3

### Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow', region='ap-  
south-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8p.2- south-1.amazonaws.com/ sagemaker-rl-ray- container:ray-0.8.2- tf-<tag>		training	CPU, GPU	py36
462105765813.dkr.e0.8p.5- south-1.amazonaws.com/ sagemaker-rl-ray- container:ray-0.8.5- tf-<tag>		training	CPU, GPU	py36
520713654638.dkr.e0.5p- south-1.amazonaws.com/ sagemaker-rl- tensorflow:ray0.5- <tag>		training	CPU, GPU	py3
520713654638.dkr.e0.5p.3- south-1.amazonaws.com/ sagemaker-rl- tensorflow:ray0.5.3- <tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6p- south-1.amazonaws.com/ sagemaker-rl- tensorflow:ray0.6- <tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6p.5- south-1.amazonaws.com/ sagemaker-rl- tensorflow:ray0.6.5- <tag>		training	CPU, GPU	py3

## VW (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='vw', region='ap-south-1', version='8.7.0', image_scope='training')
```

Registry path	Version	Job types (image scope)		
462105765813.dkr.ecr.ap-8.7.0.south-1.amazonaws.com/sagemaker-rl-vw-container:vw-8.7.0-<tag>		training		

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost', region='ap-south-1', version='1.2-1')
```

Registry path	Version	Job types (image scope)		
720646828776.dkr.ecr.ap-0.90-1.south-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
720646828776.dkr.ecr.ap-0.90-2.south-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
720646828776.dkr.ecr.ap-1.0-1.south-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
720646828776.dkr.ecr.ap-1.2-1.south-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
720646828776.dkr.ecr.ap-1.2-2.south-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
720646828776.dkr.ecr.ap-1.3-1.south-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		

Registry path	Version	Job types (image scope)		
991648021394.dkr.ecr.ap-1.south-1.amazonaws.com/xgboost:<tag>		inference, training		

## Docker Registry Paths and Example Code for Asia Pacific (Seoul) (ap-northeast-2)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

### Topics

- [BlazingText \(algorithm\) \(p. 920\)](#)
- [Chainer \(DLC\) \(p. 920\)](#)
- [Clarify \(algorithm\) \(p. 920\)](#)
- [Data Wrangler \(algorithm\) \(p. 921\)](#)
- [Debugger \(algorithm\) \(p. 921\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 921\)](#)
- [Factorization Machines \(algorithm\) \(p. 922\)](#)
- [Hugging Face \(algorithm\) \(p. 922\)](#)
- [IP Insights \(algorithm\) \(p. 923\)](#)
- [Image classification \(algorithm\) \(p. 923\)](#)
- [Inferentia MXNet \(DLC\) \(p. 923\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 924\)](#)
- [K-Means \(algorithm\) \(p. 924\)](#)
- [KNN \(algorithm\) \(p. 924\)](#)
- [LDA \(algorithm\) \(p. 924\)](#)
- [Linear Learner \(algorithm\) \(p. 925\)](#)
- [MXNet \(DLC\) \(p. 925\)](#)
- [MXNet Coach \(DLC\) \(p. 928\)](#)
- [Model Monitor \(algorithm\) \(p. 928\)](#)
- [NTM \(algorithm\) \(p. 928\)](#)
- [Neo Image Classification \(algorithm\) \(p. 929\)](#)
- [Neo MXNet \(DLC\) \(p. 929\)](#)
- [Neo PyTorch \(DLC\) \(p. 929\)](#)
- [Neo Tensorflow \(DLC\) \(p. 930\)](#)
- [Neo XGBoost \(algorithm\) \(p. 930\)](#)
- [Object Detection \(algorithm\) \(p. 931\)](#)
- [Object2Vec \(algorithm\) \(p. 931\)](#)
- [PCA \(algorithm\) \(p. 931\)](#)
- [PyTorch \(DLC\) \(p. 931\)](#)
- [Random Cut Forest \(algorithm\) \(p. 934\)](#)
- [Ray PyTorch \(DLC\) \(p. 934\)](#)
- [Scikit-learn \(algorithm\) \(p. 934\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 935\)](#)
- [Seq2Seq \(algorithm\) \(p. 935\)](#)

- [Spark \(algorithm\) \(p. 935\)](#)
- [SparkML Serving \(algorithm\) \(p. 936\)](#)
- [Tensorflow \(DLC\) \(p. 936\)](#)
- [Tensorflow Coach \(DLC\) \(p. 943\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 944\)](#)
- [Tensorflow Ray \(DLC\) \(p. 944\)](#)
- [VW \(algorithm\) \(p. 945\)](#)
- [XGBoost \(algorithm\) \(p. 946\)](#)

### [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext',region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
306986355934.dkr.ecr.ap-1 northeast-2.amazonaws.com/ blazingtext:<tag>		inference, training		

### [Chainer \(DLC\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer',region='ap-northeast-2',version='5.0.0',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-1 northeast-2.amazonaws.com/ sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-1 northeast-2.amazonaws.com/ sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-1 northeast-2.amazonaws.com/ sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3

### [Clarify \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify',region='ap-northeast-2',version='1.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
263625296855.dkr.ecr.ap-1.0 northeast-2.amazonaws.com/ sagemaker-clarify- processing:<tag>		processing		

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler',region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
131546521161.dkr.ecr.ap-1.x northeast-2.amazonaws.com/ sagemaker- data-wrangler- container:<tag>		processing		

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger',region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
578805364391.dkr.ecr.ap-latest northeast-2.amazonaws.com/ sagemaker-debugger- rules:<tag>		debugger		

### DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar',region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
204372634319.dkr.ecr.ap-1 northeast-2.amazonaws.com/ forecasting- deepar:<tag>		inference, training		

### Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines',region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
835164637446.dkr.ecr.ap-1 northeast-2.amazonaws.com/ factorization- machines:<tag>		inference, training		

### Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface',region='ap-  
northeast-2',version='4.4.2',image_scope='training',base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.ap-4.4.2 northeast-2.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.ap-4.5.0 northeast-2.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.ap-4.4.2 northeast-2.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		
763104351884.dkr.ecr.ap-4.5.0 northeast-2.amazonaws.com/ huggingface-		training		

Registry path	Version	Job types (image scope)		
tensorflow-training:<tag>				

### IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights',region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
835164637446.dkr.ecr.ap-1 northeast-2.amazonaws.com/ ipinsights:<tag>		inference, training		

### Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification',region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
306986355934.dkr.ecr.ap-1 northeast-2.amazonaws.com/ image-classification:<tag>		inference, training		

### Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='ap-northeast-2',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
151534178276.dkr.ecr.ap-1 northeast-2.amazonaws.com/ sagemaker-neo-mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='ap-northeast-2',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
151534178276.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

## K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans',region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
835164637446.dkr.ecr.ap-1-northeast-2.amazonaws.com/kmeans:<tag>		inference, training		

## KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn',region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
835164637446.dkr.ecr.ap-1-northeast-2.amazonaws.com/knn:<tag>		inference, training		

## LDA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='lda',region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
293181348795.dkr.ecr.ap-1 northeast-2.amazonaws.com/ lda:<tag>		inference, training		

### Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
835164637446.dkr.ecr.ap-1 northeast-2.amazonaws.com/ linear-learner:<tag>		inference, training		

### MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='ap-northeast-2', version='1.4.1', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr.30- northeast-2.amazonaws.com/ sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.edr.40- northeast-2.amazonaws.com/ sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.edr.40- northeast-2.amazonaws.com/ sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.41- northeast-2.amazonaws.com/ sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.1p-1 northeast-2.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.1p-1 northeast-2.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.0p- northeast-2.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.0p- northeast-2.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.1p- northeast-2.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1p- northeast-2.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.2p- northeast-2.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.2p- northeast-2.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.3p- northeast-2.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.3p- northeast-2.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.4p- northeast-2.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <del>4.0</del> -northeast-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <del>4.0</del> -northeast-2.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <del>5.0</del> -northeast-2.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> -northeast-2.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr. <del>4.0</del> -northeast-2.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>6.0</del> -northeast-2.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> -northeast-2.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>8.0</del> -northeast-2.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37
763104351884.dkr.edr. <del>4.0</del> -northeast-2.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>6.0</del> -northeast-2.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> -northeast-2.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.ap-northeast-2.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

### MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='ap-northeast-2', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor', region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
709848358524.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='ntm', region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
835164637446.dkr.ecr.ap-1 northeast-2.amazonaws.com/ ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo', region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
151534178276.dkr.ecr.ap-latest northeast-2.amazonaws.com/ image-classification- neo:<tag>		inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet', region='ap-northeast-2', version='1.8', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
151534178276.dkr.ecr.ap- northeast-2.amazonaws.com/ sagemaker- inference- mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='ap-northeast-2', version='1.6', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
151534178276.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
151534178276.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
151534178276.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='ap-northeast-2', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
151534178276.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

### Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
151534178276.dkr.ecr.ap-latest-northeast-2.amazonaws.com/xgboost-neo:<tag>		inference		

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection',region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
306986355934.dkr.ecr.ap-1 northeast-2.amazonaws.com/ object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec',region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
835164637446.dkr.ecr.ap-1 northeast-2.amazonaws.com/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca',region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
835164637446.dkr.ecr.ap-1 northeast-2.amazonaws.com/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch',region='ap-northeast-2',version='1.8.0',py_version='py3',image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.40-northeast-2.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.40-northeast-2.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0-northeast-2.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0-northeast-2.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0-northeast-2.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0-northeast-2.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.3.0-northeast-2.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr.3.0-northeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr.3.0-northeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr.4.0-northeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr.5.0-northeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <del>6.0</del> -northeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>7.0</del> -northeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>8.0</del> -northeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>8.1</del> -northeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>2.0</del> -northeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>3.0</del> -northeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>4.0</del> -northeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>5.0</del> -northeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>6.0</del> -northeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>7.0</del> -northeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>8.0</del> -northeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.ap-northeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

### Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
835164637446.dkr.ecr.ap-1-northeast-2.amazonaws.com/randomcutforest:<tag>		inference, training		

### Ray PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-pytorch', region='ap-northeast-2', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.ecr.ap-0.8.5-northeast-2.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-torch-<tag>		training	CPU, GPU	py36

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn', region='ap-northeast-2', version='0.23-1', image_scope='inference')
```

Registry path	Version	Job types (image scope)		
366743142698.dkr.ecr.ap-0.20.0-northeast-2.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
sagemaker-scikit-learn:<tag>				
366743142698.dkr.ecr.ap-0.23-1 northeast-2.amazonaws.com/ sagemaker-scikit-learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation', region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
306986355934.dkr.ecr.ap-1 northeast-2.amazonaws.com/ semantic-segmentation:<tag>		inference, training		

### Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq', region='ap-northeast-2')
```

Registry path	Version	Job types (image scope)		
306986355934.dkr.ecr.ap-1 northeast-2.amazonaws.com/ seq2seq:<tag>		inference, training		

### Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark', region='ap-northeast-2', version='3.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
860869212795.dkr.ecr.ap-2.4 northeast-2.amazonaws.com/		processing		

Registry path	Version	Job types (image scope)		
sagemaker-spark-processing:<tag>				
860869212795.dkr.ecr.ap-3.0 northeast-2.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving', region='ap-northeast-2', version='2.4')
```

Registry path	Version	Job types (image scope)		
366743142698.dkr.ecr.ap-2.2 northeast-2.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		
366743142698.dkr.ecr.ap-2.4 northeast-2.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='ap-northeast-2', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-0.0 northeast-2.amazonaws.com/ sagemaker-tensorflow-eia:<tag>		eia	CPU	py2
520713654638.dkr.ecr.ap-0.0 northeast-2.amazonaws.com/ sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e-dr.10.0 northeast-2.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e-dr.10.1 northeast-2.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
520713654638.dkr.e-dr.10.0 northeast-2.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.10.0 northeast-2.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.10.0 northeast-2.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.10.0 northeast-2.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.10.0 northeast-2.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.10.0 northeast-2.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.10.0 northeast-2.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-tensorflow:<tag>	4.10	inference	CPU, GPU	py2
520713654638.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-tensorflow:<tag>	4.10	training	CPU, GPU	py2
520713654638.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-tensorflow:<tag>	5.00	inference	CPU, GPU	py2
520713654638.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-tensorflow:<tag>	5.00	training	CPU, GPU	py2
520713654638.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-tensorflow:<tag>	6.00	inference	CPU, GPU	py2
520713654638.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-tensorflow:<tag>	6.00	training	CPU, GPU	py2
520713654638.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-tensorflow:<tag>	7.00	inference	CPU, GPU	py2
520713654638.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-tensorflow:<tag>	7.00	training	CPU, GPU	py2
520713654638.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-tensorflow:<tag>	8.00	inference	CPU, GPU	py2
520713654638.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-tensorflow:<tag>	8.00	training	CPU, GPU	py2
520713654638.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-tensorflow:<tag>	9.00	inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>9.0</u> -northeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.e <u>10.0</u> -northeast-2.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>10.0</u> -northeast-2.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>10.0</u> -northeast-2.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>10.0</u> -northeast-2.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>10.0</u> -northeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>10.0</u> -northeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>10.0</u> -northeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>10.2</u> -northeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>10.3</u> -northeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.15</u> .4 northeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.15</u> .5 northeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.16</u> - northeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.16</u> - northeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.17</u> - northeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.17</u> - northeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.17</u> - northeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.18</u> - northeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.19</u> - northeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.19</u> - northeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.19</u> - northeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <del>0.20</del> . <sup>20</sup> -northeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.21</del> . <sup>21</sup> -northeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.22</del> . <sup>22</sup> -northeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.30</del> . <sup>30</sup> -northeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.31</del> . <sup>31</sup> -northeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.32</del> . <sup>32</sup> -northeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.41</del> . <sup>41</sup> -northeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.edr. <del>14</del> . <sup>14</sup> -northeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>14</del> . <sup>14</sup> .0-northeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>15</del> . <sup>15</sup> .0-northeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>15</del> . <sup>15</sup> .2-northeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.15.3</u> northeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.e <u>0.15.4</u> northeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0.15.5</u> northeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0.16-</u> northeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.16-</u> northeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.16-</u> northeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.16.3</u> northeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.16.4</u> northeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.16.10-</u> northeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.16.12-</u> northeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e0.1p-northeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e0.2p-northeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.2p-northeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.2p-northeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.3p-northeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.3p-northeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.3p-northeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.4p-northeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow',region='ap-northeast-2',version='1.0.0',image_scope='training',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.0p-northeast-2.amazonaws.com/sagemaker-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
rl-coach-container:coach-1.0.0-tf-<tag>				
520713654638.dkr.e0.10-northeast-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10.1-northeast-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11-northeast-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.0-northeast-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.1-northeast-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inferentia (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow',region='ap-northeast-2',version='1.15.0',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
151534178276.dkr.e0.15.0-northeast-2.amazonaws.com/sagemaker-neo-tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow', region='ap-northeast-2', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8.2-northeast-2.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.2-tf-<tag>	e0.8.2-	training	CPU, GPU	py36
462105765813.dkr.e0.8.5-northeast-2.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-tf-<tag>	e0.8.5-	training	CPU, GPU	py36
520713654638.dkr.e0.5.ap-northeast-2.amazonaws.com/sagemaker-rl-tensorflow:ray0.5-<tag>	e0.5.ap-	training	CPU, GPU	py3
520713654638.dkr.e0.5.ap-northeast-2.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3-<tag>	e0.5.ap-	training	CPU, GPU	py3
520713654638.dkr.e0.6.ap-northeast-2.amazonaws.com/sagemaker-rl-tensorflow:ray0.6-<tag>	e0.6.ap-	training	CPU, GPU	py3
520713654638.dkr.e0.6.ap-northeast-2.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5-<tag>	e0.6.ap-	training	CPU, GPU	py3

## VW (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='vw', region='ap-northeast-2', version='8.7.0', image_scope='training')
```

Registry path	Version	Job types (image scope)		
462105765813.dkr.ecr.ap-8.7.0-northeast-2.amazonaws.com/		training		

Registry path	Version	Job types (image scope)		
sagemaker-rl-vw-container:vw-8.7.0-<tag>				

### XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost',region='ap-northeast-2',version='1.2-1')
```

Registry path	Version	Job types (image scope)		
306986355934.dkr.ecr.ap-1-northeast-2.amazonaws.com/xgboost:<tag>		inference, training		
366743142698.dkr.ecr.ap-0.90-1-northeast-2.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
366743142698.dkr.ecr.ap-0.90-2-northeast-2.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
366743142698.dkr.ecr.ap-1.0-1-northeast-2.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
366743142698.dkr.ecr.ap-1.2-1-northeast-2.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
366743142698.dkr.ecr.ap-1.2-2-northeast-2.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
366743142698.dkr.ecr.ap-1.3-1-northeast-2.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		

### Docker Registry Paths and Example Code for Asia Pacific (Singapore) (ap-southeast-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

## Topics

- [BlazingText \(algorithm\) \(p. 948\)](#)
- [Chainer \(DLC\) \(p. 948\)](#)
- [Clarify \(algorithm\) \(p. 948\)](#)
- [Data Wrangler \(algorithm\) \(p. 949\)](#)
- [Debugger \(algorithm\) \(p. 949\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 949\)](#)
- [Factorization Machines \(algorithm\) \(p. 950\)](#)
- [Hugging Face \(algorithm\) \(p. 950\)](#)
- [IP Insights \(algorithm\) \(p. 951\)](#)
- [Image classification \(algorithm\) \(p. 951\)](#)
- [Inferentia MXNet \(DLC\) \(p. 951\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 951\)](#)
- [K-Means \(algorithm\) \(p. 952\)](#)
- [KNN \(algorithm\) \(p. 952\)](#)
- [LDA \(algorithm\) \(p. 952\)](#)
- [Linear Learner \(algorithm\) \(p. 953\)](#)
- [MXNet \(DLC\) \(p. 953\)](#)
- [MXNet Coach \(DLC\) \(p. 955\)](#)
- [Model Monitor \(algorithm\) \(p. 956\)](#)
- [NTM \(algorithm\) \(p. 956\)](#)
- [Neo Image Classification \(algorithm\) \(p. 956\)](#)
- [Neo MXNet \(DLC\) \(p. 957\)](#)
- [Neo PyTorch \(DLC\) \(p. 957\)](#)
- [Neo Tensorflow \(DLC\) \(p. 958\)](#)
- [Neo XGBoost \(algorithm\) \(p. 958\)](#)
- [Object Detection \(algorithm\) \(p. 958\)](#)
- [Object2Vec \(algorithm\) \(p. 959\)](#)
- [PCA \(algorithm\) \(p. 959\)](#)
- [PyTorch \(DLC\) \(p. 959\)](#)
- [Random Cut Forest \(algorithm\) \(p. 961\)](#)
- [Ray PyTorch \(DLC\) \(p. 962\)](#)
- [Scikit-learn \(algorithm\) \(p. 962\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 962\)](#)
- [Seq2Seq \(algorithm\) \(p. 963\)](#)
- [Spark \(algorithm\) \(p. 963\)](#)
- [SparkML Serving \(algorithm\) \(p. 963\)](#)
- [Tensorflow \(DLC\) \(p. 964\)](#)
- [Tensorflow Coach \(DLC\) \(p. 971\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 972\)](#)
- [Tensorflow Ray \(DLC\) \(p. 972\)](#)
- [VW \(algorithm\) \(p. 973\)](#)
- [XGBoost \(algorithm\) \(p. 973\)](#)

## BlazingText (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext', region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1.southeast-1.amazonaws.com/blazingtext:<tag>		inference, training		

## Chainer (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer', region='ap-southeast-1', version='5.0.0', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-1.southeast-1.amazonaws.com/sagemaker-chainer:<tag>	4.10-1	inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-1.southeast-1.amazonaws.com/sagemaker-chainer:<tag>	4.10-2	inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-1.southeast-1.amazonaws.com/sagemaker-chainer:<tag>	5.0.0-1	inference, training	CPU, GPU	py2, py3

## Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify', region='ap-southeast-1', version='1.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
834264404009.dkr.ecr.ap-1.0.southeast-1.amazonaws.com/		processing		

Registry path	Version	Job types (image scope)		
sagemaker-clarify-processing:<tag>				

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler', region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
119527597002.dkr.ecr.ap-1.x southeast-1.amazonaws.com/ sagemaker- data-wrangler- container:<tag>		processing		

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger', region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
972752614525.dkr.ecr.ap-latest southeast-1.amazonaws.com/ sagemaker-debugger- rules:<tag>		debugger		

### DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar', region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1 southeast-1.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
forecasting-deepar:<tag>				

### Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines', region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1.southeast-1.amazonaws.com/factorization-machines:<tag>		inference, training		

### Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='ap-southeast-1', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.ap-4.4.2.southeast-1.amazonaws.com/huggingface-pytorch-training:<tag>		training		
763104351884.dkr.ecr.ap-4.5.0.southeast-1.amazonaws.com/huggingface-pytorch-training:<tag>		training		
763104351884.dkr.ecr.ap-4.4.2.southeast-1.amazonaws.com/huggingface-tensorflow-training:<tag>		training		
763104351884.dkr.ecr.ap-4.5.0.southeast-1.amazonaws.com/huggingface-tensorflow-training:<tag>		training		

## IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights',region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1 southeast-1.amazonaws.com/ ipinsights:<tag>		inference, training		

## Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification',region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1 southeast-1.amazonaws.com/ image- classification:<tag>		inference, training		

## Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='ap-  
southeast-1',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
324986816169.dkr.ecr.ap- southeast-1.amazonaws.com/ sagemaker-neo- mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='ap-  
southeast-1',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
324986816169.dkr.ecr.ap-southeast-1.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

### K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans', region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1-southeast-1.amazonaws.com/kmeans:<tag>		inference, training		

### KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn', region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1-southeast-1.amazonaws.com/knn:<tag>		inference, training		

### LDA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='lda', region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1-southeast-1.amazonaws.com/lda:<tag>		inference, training		

## Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1.southeast-1.amazonaws.com/linear-learner:<tag>		inference, training		

## MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='ap-southeast-1', version='1.4.1', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr.3.0-southeast-1.amazonaws.com/sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.edr.4.0-southeast-1.amazonaws.com/sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.edr.4.0-southeast-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.4.1-southeast-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e0.10.1-southeast-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.10.1-southeast-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <b>00</b> -southeast-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>00</b> -southeast-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>10</b> -southeast-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>10</b> -southeast-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>20</b> -southeast-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>20</b> -southeast-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>30</b> -southeast-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>30</b> -southeast-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>40</b> -southeast-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>40</b> -southeast-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <b>41</b> -southeast-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <del>5.0</del> -southeast-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> -southeast-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr. <del>4.0</del> -southeast-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>6.0</del> -southeast-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> -southeast-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>2.0</del> -southeast-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37
763104351884.dkr.edr. <del>4.0</del> -southeast-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>6.0</del> -southeast-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> -southeast-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>2.0</del> -southeast-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

## MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='coach-mxnet',region='ap-southeast-1',version='0.11',py_version='py3',image_scope='training',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.1p-southeast-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.1p.0southeast-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor',region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
245545462676.dkr.ecr.ap-southeast-1.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm',region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1southeast-1.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='image-classification-neo',region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
324986816169.dkr.ecr.ap-southeast-1.amazonaws.com/image-classification-neo:<tag>		inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet',region='ap-southeast-1',version='1.8',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
324986816169.dkr.ecr.ap-southeast-1.amazonaws.com/sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch',region='ap-southeast-1',version='1.6',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
324986816169.dkr.ecr.ap-southeast-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
324986816169.dkr.ecr.ap-southeast-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
324986816169.dkr.ecr.ap-southeast-1.amazonaws.com/		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-inference-pytorch:<tag>				

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='ap-southeast-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
324986816169.dkr.ecr.ap-15.3.southeast-1.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

### Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
324986816169.dkr.ecr.ap-latest.southeast-1.amazonaws.com/xgboost-neo:<tag>		inference		

### Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1.southeast-1.amazonaws.com/object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec',region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1 southeast-1.amazonaws.com/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca',region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1 southeast-1.amazonaws.com/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch',region='ap-southeast-1',version='1.8.0',py_version='py3',image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.40- southeast-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.40- southeast-1.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e0.40- southeast-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr.00-southeast-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.10-southeast-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.10-southeast-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.30-southeast-1.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr.20-southeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr.30-southeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr.40-southeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr.50-southeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr.60-southeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr.70-southeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr.80-southeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr.ap-southeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr.ap-southeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.ap-southeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.ap-southeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.ap-southeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr.ap-southeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr.ap-southeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr.ap-southeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr.ap-southeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

## Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1.southeast-1.amazonaws.com/randomcutforest:<tag>		inference, training		

### Ray PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-pytorch', region='ap-southeast-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.ecr.ap-0.8.5-southeast-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-torch-<tag>		training	CPU, GPU	py36

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn', region='ap-southeast-1', version='0.23-1', image_scope='inference')
```

Registry path	Version	Job types (image scope)		
121021644041.dkr.ecr.ap-0.20.0-southeast-1.amazonaws.com/sagemaker-scikit-learn:<tag>		inference, training		
121021644041.dkr.ecr.ap-0.23-1-southeast-1.amazonaws.com/sagemaker-scikit-learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='semantic-segmentation',region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1.southeast-1.amazonaws.com/semantic-segmentation:<tag>		inference, training		

## Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='ap-southeast-1')
```

Registry path	Version	Job types (image scope)		
475088953585.dkr.ecr.ap-1.southeast-1.amazonaws.com/seq2seq:<tag>		inference, training		

## Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='ap-southeast-1',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
759080221371.dkr.ecr.ap-2.4.southeast-1.amazonaws.com/sagemaker-spark-processing:<tag>		processing		
759080221371.dkr.ecr.ap-3.0.southeast-1.amazonaws.com/sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving',region='ap-southeast-1',version='2.4')
```

Registry path	Version	Job types (image scope)		
121021644041.dkr.ecr.ap-2.2 southeast-1.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		
121021644041.dkr.ecr.ap-2.4 southeast-1.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		

### Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='ap-southeast-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-10.0 southeast-1.amazonaws.com/ sagemaker- tensorflow- eia:<tag>		eia	CPU	py2
520713654638.dkr.ecr.ap-0 southeast-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-0 southeast-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-1 southeast-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr.ap-0 southeast-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.ecr.ap-0 southeast-1.amazonaws.com/		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-tensorflow-serving-eia:<tag>				
520713654638.dkr.edr.1p0-southeast-1.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>		eia	CPU	-
520713654638.dkr.edr.1p0-southeast-1.amazonaws.com/sagemaker-tensorflow-serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.edr.1p0-southeast-1.amazonaws.com/sagemaker-tensorflow-serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.edr.1p0-southeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr.1p0-southeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr.4p-southeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr.4p-southeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr.5p0-southeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr.5p0-southeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr.6p0-southeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <del>6.0</del> -southeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> -southeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> -southeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> -southeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> -southeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>9.0</del> -southeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>9.0</del> -southeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <del>10.0</del> -southeast-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <del>15.0</del> -southeast-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <del>20.0</del> -southeast-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <del>2.3</del> .0-southeast-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <del>2.3</del> .0-southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>2.3</del> .0-southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>2.3</del> .0-southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>2.3</del> .2-southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>2.3</del> .3-southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>2.3</del> .4-southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>2.3</del> .5-southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>2.3</del> .6-southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>2.3</del> .7-southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>2.3</del> .8-southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <del>0.03</del> -southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.04</del> -southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.10</del> -southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.11</del> -southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.12</del> -southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.13</del> -southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.20</del> -southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.21</del> -southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.22</del> -southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.23</del> -southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.30</del> -southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.31</del> -southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <sup>1</sup> . <sub>2</sub> -southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.edr. <sup>1</sup> . <sub>4</sub> -southeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.edr. <sup>1</sup> . <sub>5</sub> .1-southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <sup>1</sup> . <sub>6</sub> .0-southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <sup>1</sup> . <sub>6</sub> .0-southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <sup>1</sup> . <sub>6</sub> .2-southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.edr. <sup>1</sup> . <sub>6</sub> .3-southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.edr. <sup>1</sup> . <sub>6</sub> .4-southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.edr. <sup>1</sup> . <sub>6</sub> .5-southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.edr. <sup>2</sup> . <sub>0</sub> .0-southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <sup>2</sup> . <sub>0</sub> .1-southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.02</u> -southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.03</u> -southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.04</u> -southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.10</u> -southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.11</u> -southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.12</u> -southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.13</u> -southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.20</u> -southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.21</u> -southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.22</u> -southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.30</u> -southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e0.31-p-southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.31-p-southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.41-p-southeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow', region='ap-southeast-1', version='1.0.0', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.0-p-southeast-1.amazonaws.com/sagemaker-rl-coach-container:coach-1.0.0-tf-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10-p-southeast-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10.1-p-southeast-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11-p-southeast-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.0-p-southeast-1.amazonaws.com/		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-rl-tensorflow:coach0.11.0-<tag>				
520713654638.dkr.e0.1p.1 southeast-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inference (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow',region='ap-southeast-1',version='1.15.0',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
324986816169.dkr.e0.1p.0 southeast-1.amazonaws.com/sagemaker-neo-tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow',region='ap-southeast-1',version='0.8.5',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8p.2-southeast-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.2-tf-<tag>		training	CPU, GPU	py36
462105765813.dkr.e0.8p.5-southeast-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-tf-<tag>		training	CPU, GPU	py36
520713654638.dkr.e0.5p-southeast-1.amazonaws.com/		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-rl-tensorflow:ray0.5- <i>&lt;tag&gt;</i>				
520713654638.dkr.e0.5.3-southeast-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3- <i>&lt;tag&gt;</i>		training	CPU, GPU	py3
520713654638.dkr.e0.6-ap-southeast-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6- <i>&lt;tag&gt;</i>		training	CPU, GPU	py3
520713654638.dkr.e0.6.5-southeast-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5- <i>&lt;tag&gt;</i>		training	CPU, GPU	py3

## VW (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='vw', region='ap-southeast-1', version='8.7.0', image_scope='training')
```

Registry path	Version	Job types (image scope)		
462105765813.dkr.ecr.ap-8.7.0-southeast-1.amazonaws.com/sagemaker-rl-vw-container:vw-8.7.0- <i>&lt;tag&gt;</i>		training		

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost', region='ap-southeast-1', version='1.2-1')
```

Registry path	Version	Job types (image scope)		
121021644041.dkr.ecr.ap-0.90-1-southeast-1.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
sagemaker-xgboost:<tag>				
121021644041.dkr.ecr.ap-0.90-2 southeast-1.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
121021644041.dkr.ecr.ap-1.0-1 southeast-1.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
121021644041.dkr.ecr.ap-1.2-1 southeast-1.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
121021644041.dkr.ecr.ap-1.2-2 southeast-1.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
121021644041.dkr.ecr.ap-1.3-1 southeast-1.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
475088953585.dkr.ecr.ap-1 southeast-1.amazonaws.com/ xgboost:<tag>		inference, training		

## Docker Registry Paths and Example Code for Asia Pacific (Sydney) (ap-southeast-2)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

### Topics

- [BlazingText \(algorithm\) \(p. 975\)](#)
- [Chainer \(DLC\) \(p. 975\)](#)
- [Clarify \(algorithm\) \(p. 976\)](#)
- [Data Wrangler \(algorithm\) \(p. 976\)](#)
- [Debugger \(algorithm\) \(p. 977\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 977\)](#)
- [Factorization Machines \(algorithm\) \(p. 977\)](#)
- [Hugging Face \(algorithm\) \(p. 977\)](#)
- [IP Insights \(algorithm\) \(p. 978\)](#)
- [Image classification \(algorithm\) \(p. 978\)](#)
- [Inferentia MXNet \(DLC\) \(p. 979\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 979\)](#)
- [K-Means \(algorithm\) \(p. 979\)](#)

- [KNN \(algorithm\) \(p. 980\)](#)
- [LDA \(algorithm\) \(p. 980\)](#)
- [Linear Learner \(algorithm\) \(p. 980\)](#)
- [MXNet \(DLC\) \(p. 980\)](#)
- [MXNet Coach \(DLC\) \(p. 983\)](#)
- [Model Monitor \(algorithm\) \(p. 983\)](#)
- [NTM \(algorithm\) \(p. 984\)](#)
- [Neo Image Classification \(algorithm\) \(p. 984\)](#)
- [Neo MXNet \(DLC\) \(p. 984\)](#)
- [Neo PyTorch \(DLC\) \(p. 985\)](#)
- [Neo Tensorflow \(DLC\) \(p. 985\)](#)
- [Neo XGBoost \(algorithm\) \(p. 986\)](#)
- [Object Detection \(algorithm\) \(p. 986\)](#)
- [Object2Vec \(algorithm\) \(p. 986\)](#)
- [PCA \(algorithm\) \(p. 986\)](#)
- [PyTorch \(DLC\) \(p. 987\)](#)
- [Random Cut Forest \(algorithm\) \(p. 989\)](#)
- [Ray PyTorch \(DLC\) \(p. 989\)](#)
- [Scikit-learn \(algorithm\) \(p. 990\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 990\)](#)
- [Seq2Seq \(algorithm\) \(p. 990\)](#)
- [Spark \(algorithm\) \(p. 991\)](#)
- [SparkML Serving \(algorithm\) \(p. 991\)](#)
- [Tensorflow \(DLC\) \(p. 991\)](#)
- [Tensorflow Coach \(DLC\) \(p. 999\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 999\)](#)
- [Tensorflow Ray \(DLC\) \(p. 1000\)](#)
- [VW \(algorithm\) \(p. 1001\)](#)
- [XGBoost \(algorithm\) \(p. 1001\)](#)

### BlazingText (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext', region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
544295431143.dkr.ecr.ap-1.southeast-2.amazonaws.com/blazingtext:<tag>		inference, training		

### Chainer (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer',region='ap-southeast-2',version='5.0.0',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-chainer:<tag>	4.10.0	inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-chainer:<tag>	4.11.0	inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-chainer:<tag>	5.0.0	inference, training	CPU, GPU	py2, py3

### Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify',region='ap-southeast-2',version='1.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
007051062584.dkr.ecr.ap-1.0-southeast-2.amazonaws.com/sagemaker-clarify-processing:<tag>		processing		

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler',region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
422173101802.dkr.ecr.ap-1.x-southeast-2.amazonaws.com/sagemaker-data-wrangler-container:<tag>		processing		

## Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger',region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
184798709955.dkr.ecr.ap-latest southeast-2.amazonaws.com/ sagemaker-debugger- rules:<tag>		debugger		

## DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar',region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
514117268639.dkr.ecr.ap-1 southeast-2.amazonaws.com/ forecasting- deepar:<tag>		inference, training		

## Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines',region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
712309505854.dkr.ecr.ap-1 southeast-2.amazonaws.com/ factorization- machines:<tag>		inference, training		

## Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='huggingface',region='ap-southeast-2',version='4.4.2',image_scope='training',base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.ap-southeast-2.amazonaws.com/huggingface-pytorch-training:<tag>	4.4.2	training		
763104351884.dkr.ecr.ap-southeast-2.amazonaws.com/huggingface-pytorch-training:<tag>	4.5.0	training		
763104351884.dkr.ecr.ap-southeast-2.amazonaws.com/huggingface-tensorflow-training:<tag>	4.4.2	training		
763104351884.dkr.ecr.ap-southeast-2.amazonaws.com/huggingface-tensorflow-training:<tag>	4.5.0	training		

### IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights',region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
712309505854.dkr.ecr.ap-southeast-2.amazonaws.com/ipinsights:<tag>	1	inference, training		

### Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification',region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
544295431143.dkr.ecr.ap-southeast-2.amazonaws.com/	1	inference, training		

Registry path	Version	Job types (image scope)		
image-classification:<tag>				

### Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='ap-southeast-2',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
355873309152.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-neo-mxnet:<tag>		inference	inf	py3

### Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='ap-southeast-2',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
355873309152.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

### K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans',region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
712309505854.dkr.ecr.ap-1-southeast-2.amazonaws.com/kmeans:<tag>		inference, training		

## KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn', region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
712309505854.dkr.ecr.ap-1 southeast-2.amazonaws.com/ knn:<tag>		inference, training		

## LDA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='lda', region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
297031611018.dkr.ecr.ap-1 southeast-2.amazonaws.com/ lda:<tag>		inference, training		

## Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
712309505854.dkr.ecr.ap-1 southeast-2.amazonaws.com/ linear-learner:<tag>		inference, training		

## MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='ap-southeast-2', version='1.4.1', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <sup>ap</sup> -southeast-2.amazonaws.com/sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.edr. <sup>ap</sup> -southeast-2.amazonaws.com/sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.edr. <sup>ap</sup> -southeast-2.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <sup>ap</sup> -southeast-2.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e0. <sup>ap</sup> .1-southeast-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0. <sup>ap</sup> .1-southeast-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <sup>ap</sup> -southeast-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <sup>ap</sup> -southeast-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <sup>ap</sup> -southeast-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <sup>ap</sup> -southeast-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <sup>ap</sup> -southeast-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr.2p-southeast-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.3p-southeast-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.3p-southeast-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.4p-southeast-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.4p-southeast-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr.4p-southeast-2.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr.5p-southeast-2.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr.7p-southeast-2.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr.4p-southeast-2.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr.6p-southeast-2.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr.7p-southeast-2.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr.ap-southeast-2.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37
763104351884.dkr.edr.ap-southeast-2.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr.ap-southeast-2.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.ap-southeast-2.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr.ap-southeast-2.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

## MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='ap-southeast-2', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr.ap-southeast-2.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.edr.ap-southeast-2.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

## Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='model-monitor',region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
563025443158.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm',region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
712309505854.dkr.ecr.ap-1-southeast-2.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo',region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
355873309152.dkr.ecr.ap-latest-southeast-2.amazonaws.com/image-classification-neo:<tag>		inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet',region='ap-southeast-2',version='1.8',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
355873309152.dkr.e-dr.ap-southeast-2.amazonaws.com/sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='ap-southeast-2', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
355873309152.dkr.e-dr.ap-southeast-2.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
355873309152.dkr.e-dr.ap-southeast-2.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
355873309152.dkr.e-dr.ap-southeast-2.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='ap-southeast-2', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
355873309152.dkr.e-dr.ap.3.southeast-2.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

## Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
355873309152.dkr.ecr.ap-southeast-2.amazonaws.com/xgboost-neo:<tag>		inference		

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
544295431143.dkr.ecr.ap-southeast-2.amazonaws.com/object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec', region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
712309505854.dkr.ecr.ap-southeast-2.amazonaws.com/object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca', region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
712309505854.dkr.ecr.ap-1.southeast-2.amazonaws.com/pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch', region='ap-southeast-2', version='1.8.0', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.40-southeast-2.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.40-southeast-2.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0-southeast-2.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0-southeast-2.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0-southeast-2.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0-southeast-2.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.3.0-southeast-2.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr.2.0-southeast-2.amazonaws.com/		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
pytorch-inference:<tag>				
763104351884.dkr.edr. <del>3.4</del> -southeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>4.0</del> -southeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>5.0</del> -southeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>6.0</del> -southeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>7.0</del> -southeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>8.0</del> -southeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>8.1</del> -southeast-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>2.0</del> -southeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>3.0</del> -southeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>4.0</del> -southeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.ap-southeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.ecr.ap-southeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr.ap-southeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr.ap-southeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr.ap-southeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

## Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
712309505854.dkr.ecr.ap-southeast-2.amazonaws.com/randomcutforest:<tag>		inference, training		

## Ray PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-pytorch', region='ap-southeast-2', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.ecr.ap-southeast-2.amazonaws.com/		training	CPU, GPU	py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-rl-ray-container:ray-0.8.5-torch-<tag>				

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn',region='ap-southeast-2',version='0.23-1',image_scope='inference')
```

Registry path	Version	Job types (image scope)		
783357654285.dkr.ecr.ap-0.20.0-southeast-2.amazonaws.com/sagemaker-scikit-learn:<tag>		inference, training		
783357654285.dkr.ecr.ap-0.23-1-southeast-2.amazonaws.com/sagemaker-scikit-learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
544295431143.dkr.ecr.ap-1-southeast-2.amazonaws.com/semantic-segmentation:<tag>		inference, training		

### Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='ap-southeast-2')
```

Registry path	Version	Job types (image scope)		
544295431143.dkr.ecr.ap-1 southeast-2.amazonaws.com/ seq2seq:<tag>		inference, training		

## Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark', region='ap-southeast-2', version='3.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
440695851116.dkr.ecr.ap-2.4 southeast-2.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		
440695851116.dkr.ecr.ap-3.0 southeast-2.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving', region='ap-southeast-2', version='2.4')
```

Registry path	Version	Job types (image scope)		
783357654285.dkr.ecr.ap-2.2 southeast-2.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		
783357654285.dkr.ecr.ap-2.4 southeast-2.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='tensorflow', region='ap-southeast-2', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-tensorflow-eia:<tag>	1.10.0	eia	CPU	py2
520713654638.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>	1.11.0	training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>	1.12.0	training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>	1.13.1	training	CPU, GPU	py2
520713654638.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>	1.14.0	eia	CPU	-
520713654638.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>	1.15.0	eia	CPU	-
520713654638.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>	1.16.0	eia	CPU	-
520713654638.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-tensorflow-serving:<tag>	1.17.0	inference	CPU, GPU	-
520713654638.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-	1.18.0	inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow-serving:<tag>				
520713654638.dkr.ecr. <b>10.0</b> -southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <b>10.0</b> -southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr. <b>11</b> -southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <b>11</b> -southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr. <b>10</b> -southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <b>10</b> -southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr. <b>10</b> -southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <b>10</b> -southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr. <b>10</b> -southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <b>10</b> -southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr. <b>10</b> -southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <b>10</b> -southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <sup>20</sup> .0-southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <sup>20</sup> .0-southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <sup>20</sup> .0-southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <sup>20</sup> .0-southeast-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <sup>14</sup> .0-southeast-2.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <sup>15</sup> .0-southeast-2.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <sup>16</sup> .0-southeast-2.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <sup>17</sup> .0-southeast-2.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <sup>18</sup> .0-southeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.edr. <sup>19</sup> .0-southeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>15</u> .0 southeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>15</u> .2 southeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>15</u> .3 southeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>15</u> .4 southeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>15</u> .5 southeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.0.0</u> - southeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.0.1</u> - southeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.0.2</u> - southeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.0.3</u> - southeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.0.4</u> - southeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.1.0</u> - southeast-2.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e0.1p-southeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0.1p2-southeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0.1p3-southeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0.2p0-southeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0.2p1-southeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0.2p2-southeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0.3p0-southeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0.3p1-southeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0.3p2-southeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0.4p-southeast-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.edr.1p1-southeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <del>14</del> .0 southeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>15</del> .0 southeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>15</del> .2 southeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.edr. <del>15</del> .3 southeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.edr. <del>15</del> .4 southeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.edr. <del>15</del> .5 southeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.edr. <del>16</del> .0 southeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>16</del> .1 southeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>16</del> .2 southeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>16</del> .3 southeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>16</del> .4 southeast-2.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e0.10-southeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e0.11-southeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e0.12-southeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e0.13-southeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e0.20-southeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.21-southeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.22-southeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.30-southeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.31-southeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.32-southeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.41-southeast-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

## Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow',region='ap-southeast-2',version='1.0.0',image_scope='training',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.10-southeast-2.amazonaws.com/sagemaker-rl-coach-container:coach-1.0.0-tf-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10-southeast-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10.1-southeast-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10-southeast-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10.0-southeast-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10.1-southeast-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inferentia (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow',region='ap-southeast-2',version='1.15.0',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
355873309152.dkr.e0.1p.0 southeast-2.amazonaws.com/ sagemaker-neo- tensorflow:<tag>		inference	inf	py3

### Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow',region='ap-
southeast-2',version='0.8.5',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8p- southeast-2.amazonaws.com/ sagemaker-rl-ray- container:ray-0.8.2- tf-<tag>		training	CPU, GPU	py36
462105765813.dkr.e0.8p5- southeast-2.amazonaws.com/ sagemaker-rl-ray- container:ray-0.8.5- tf-<tag>		training	CPU, GPU	py36
520713654638.dkr.e0.5p- southeast-2.amazonaws.com/ sagemaker-rl- tensorflow:ray0.5- <tag>		training	CPU, GPU	py3
520713654638.dkr.e0.5p3- southeast-2.amazonaws.com/ sagemaker-rl- tensorflow:ray0.5.3- <tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6p- southeast-2.amazonaws.com/ sagemaker-rl- tensorflow:ray0.6- <tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6p5- southeast-2.amazonaws.com/ sagemaker-rl- tensorflow:ray0.6.5- <tag>		training	CPU, GPU	py3

## VW (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='vw', region='ap-southeast-2', version='8.7.0', image_scope='training')
```

Registry path	Version	Job types (image scope)		
462105765813.dkr.ecr.ap-8.7.0.southeast-2.amazonaws.com/sagemaker-rl-vw-container:vw-8.7.0-<tag>		training		

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost', region='ap-southeast-2', version='1.2-1')
```

Registry path	Version	Job types (image scope)		
544295431143.dkr.ecr.ap-1.southeast-2.amazonaws.com/xgboost:<tag>		inference, training		
783357654285.dkr.ecr.ap-0.90-1.southeast-2.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
783357654285.dkr.ecr.ap-0.90-2.southeast-2.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
783357654285.dkr.ecr.ap-1.0-1.southeast-2.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
783357654285.dkr.ecr.ap-1.2-1.southeast-2.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
783357654285.dkr.ecr.ap-1.2-2.southeast-2.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		

Registry path	Version	Job types (image scope)		
783357654285.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		

## Docker Registry Paths and Example Code for Asia Pacific (Tokyo) (ap-northeast-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

### Topics

- [BlazingText \(algorithm\) \(p. 1003\)](#)
- [Chainer \(DLC\) \(p. 1003\)](#)
- [Clarify \(algorithm\) \(p. 1003\)](#)
- [Data Wrangler \(algorithm\) \(p. 1004\)](#)
- [Debugger \(algorithm\) \(p. 1004\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 1004\)](#)
- [Factorization Machines \(algorithm\) \(p. 1005\)](#)
- [Hugging Face \(algorithm\) \(p. 1005\)](#)
- [IP Insights \(algorithm\) \(p. 1006\)](#)
- [Image classification \(algorithm\) \(p. 1006\)](#)
- [Inferentia MXNet \(DLC\) \(p. 1006\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 1007\)](#)
- [K-Means \(algorithm\) \(p. 1007\)](#)
- [KNN \(algorithm\) \(p. 1007\)](#)
- [LDA \(algorithm\) \(p. 1007\)](#)
- [Linear Learner \(algorithm\) \(p. 1008\)](#)
- [MXNet \(DLC\) \(p. 1008\)](#)
- [MXNet Coach \(DLC\) \(p. 1011\)](#)
- [Model Monitor \(algorithm\) \(p. 1011\)](#)
- [NTM \(algorithm\) \(p. 1011\)](#)
- [Neo Image Classification \(algorithm\) \(p. 1012\)](#)
- [Neo MXNet \(DLC\) \(p. 1012\)](#)
- [Neo PyTorch \(DLC\) \(p. 1012\)](#)
- [Neo Tensorflow \(DLC\) \(p. 1013\)](#)
- [Neo XGBoost \(algorithm\) \(p. 1013\)](#)
- [Object Detection \(algorithm\) \(p. 1014\)](#)
- [Object2Vec \(algorithm\) \(p. 1014\)](#)
- [PCA \(algorithm\) \(p. 1014\)](#)
- [PyTorch \(DLC\) \(p. 1014\)](#)
- [Random Cut Forest \(algorithm\) \(p. 1017\)](#)
- [Ray PyTorch \(DLC\) \(p. 1017\)](#)
- [Scikit-learn \(algorithm\) \(p. 1017\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 1018\)](#)
- [Seq2Seq \(algorithm\) \(p. 1018\)](#)

- [Spark \(algorithm\) \(p. 1018\)](#)
- [SparkML Serving \(algorithm\) \(p. 1019\)](#)
- [Tensorflow \(DLC\) \(p. 1019\)](#)
- [Tensorflow Coach \(DLC\) \(p. 1026\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 1027\)](#)
- [Tensorflow Ray \(DLC\) \(p. 1027\)](#)
- [VW \(algorithm\) \(p. 1028\)](#)
- [XGBoost \(algorithm\) \(p. 1029\)](#)

### [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
501404015308.dkr.ecr.ap-northeast-1.amazonaws.com/blazingtext:<tag>		inference, training		

### [Chainer \(DLC\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer',region='ap-northeast-1',version='5.0.0',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-northeast-1.amazonaws.com/sagemaker-chainer:<tag>	4.1.0	inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-northeast-1.amazonaws.com/sagemaker-chainer:<tag>	4.1.0	inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-northeast-1.amazonaws.com/sagemaker-chainer:<tag>	5.0.0	inference, training	CPU, GPU	py2, py3

### [Clarify \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify',region='ap-northeast-1',version='1.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
377024640650.dkr.ecr.ap-1.0 northeast-1.amazonaws.com/ sagemaker-clarify- processing:<tag>		processing		

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
649008135260.dkr.ecr.ap-1.x northeast-1.amazonaws.com/ sagemaker- data-wrangler- container:<tag>		processing		

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
430734990657.dkr.ecr.ap-latest northeast-1.amazonaws.com/ sagemaker-debugger- rules:<tag>		debugger		

### DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
633353088612.dkr.ecr.ap-1 northeast-1.amazonaws.com/ forecasting- deepar:<tag>		inference, training		

### Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
351501993468.dkr.ecr.ap-1 northeast-1.amazonaws.com/ factorization- machines:<tag>		inference, training		

### Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface',region='ap-  
northeast-1',version='4.4.2',image_scope='training',base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.ap-4.4.2 northeast-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.ap-4.5.0 northeast-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.ap-4.4.2 northeast-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		
763104351884.dkr.ecr.ap-4.5.0 northeast-1.amazonaws.com/ huggingface-		training		

Registry path	Version	Job types (image scope)		
tensorflow-training:<tag>				

### IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
351501993468.dkr.ecr.ap-1 northeast-1.amazonaws.com/ ipinsights:<tag>		inference, training		

### Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
501404015308.dkr.ecr.ap-1 northeast-1.amazonaws.com/ image-classification:<tag>		inference, training		

### Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='ap-northeast-1',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
941853720454.dkr.ecr.ap-1 northeast-1.amazonaws.com/ sagemaker-neo-mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='ap-northeast-1',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
941853720454.dkr.ecr.ap-northeast-1.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

## K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
351501993468.dkr.ecr.ap-1-northeast-1.amazonaws.com/kmeans:<tag>		inference, training		

## KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
351501993468.dkr.ecr.ap-1-northeast-1.amazonaws.com/knn:<tag>		inference, training		

## LDA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='lda',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
258307448986.dkr.ecr.ap-1 northeast-1.amazonaws.com/ lda:<tag>		inference, training		

### Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
351501993468.dkr.ecr.ap-1 northeast-1.amazonaws.com/ linear-learner:<tag>		inference, training		

### MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='ap-northeast-1', version='1.4.1', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-1 northeast-1.amazonaws.com/ sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.ecr.ap-1 northeast-1.amazonaws.com/ sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.ecr.ap-1 northeast-1.amazonaws.com/ sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.ecr.ap-1 northeast-1.amazonaws.com/ sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.1p-1 northeast-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.1p-1 northeast-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.0p- northeast-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.0p- northeast-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.1p- northeast-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1p- northeast-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.2p- northeast-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.2p- northeast-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.3p- northeast-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.3p- northeast-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.4p- northeast-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <del>4p</del> -northeast-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <del>4p</del> -northeast-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <del>5p</del> -northeast-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <del>7p</del> -northeast-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr. <del>4p</del> -northeast-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>6p</del> -northeast-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>7p</del> -northeast-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>8p</del> -northeast-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37
763104351884.dkr.edr. <del>4p</del> -northeast-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>6p</del> -northeast-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>7p</del> -northeast-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.ap-northeast-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

### MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='ap-northeast-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-northeast-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.ecr.ap-northeast-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor', region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
574779866223.dkr.ecr.ap-northeast-1.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='ntm', region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
351501993468.dkr.ecr.ap-1 northeast-1.amazonaws.com/ ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo', region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
941853720454.dkr.ecr.ap-latest northeast-1.amazonaws.com/ image-classification- neo:<tag>		inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet', region='ap-northeast-1', version='1.8', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
941853720454.dkr.ecr.ap- northeast-1.amazonaws.com/ sagemaker- inference- mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='ap-northeast-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
941853720454.dkr.ecr.ap-northeast-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
941853720454.dkr.ecr.ap-northeast-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
941853720454.dkr.ecr.ap-northeast-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='ap-northeast-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
941853720454.dkr.ecr.ap-northeast-1.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

### Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
941853720454.dkr.ecr.ap-latest-northeast-1.amazonaws.com/xgboost-neo:<tag>		inference		

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
501404015308.dkr.ecr.ap-1 northeast-1.amazonaws.com/ object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
351501993468.dkr.ecr.ap-1 northeast-1.amazonaws.com/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
351501993468.dkr.ecr.ap-1 northeast-1.amazonaws.com/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch',region='ap-northeast-1',version='1.8.0',py_version='py3',image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.40-northeast-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.40-northeast-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0-northeast-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0-northeast-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0-northeast-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0-northeast-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.3.0-northeast-1.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr.2.0-northeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr.3.0-northeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr.4.0-northeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr.5.0-northeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <del>60</del> -northeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>71</del> -northeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>80</del> -northeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>81</del> -northeast-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>20</del> -northeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>31</del> -northeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>40</del> -northeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>50</del> -northeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>60</del> -northeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>71</del> -northeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>80</del> -northeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.ap-northeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

### Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
351501993468.dkr.ecr.ap-1northeast-1.amazonaws.com/randomcutforest:<tag>		inference, training		

### Ray PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-pytorch', region='ap-northeast-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.ecr.ap-1northeast-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-torch-<tag>		training	CPU, GPU	py36

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn', region='ap-northeast-1', version='0.23-1', image_scope='inference')
```

Registry path	Version	Job types (image scope)		
354813040037.dkr.ecr.ap-0.20.0northeast-1.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
sagemaker-scikit-learn:<tag>				
354813040037.dkr.ecr.ap-0.23-1 northeast-1.amazonaws.com/ sagemaker-scikit-learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
501404015308.dkr.ecr.ap-1 northeast-1.amazonaws.com/ semantic-segmentation:<tag>		inference, training		

### Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='ap-northeast-1')
```

Registry path	Version	Job types (image scope)		
501404015308.dkr.ecr.ap-1 northeast-1.amazonaws.com/ seq2seq:<tag>		inference, training		

### Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='ap-northeast-1',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
411782140378.dkr.ecr.ap-2.4 northeast-1.amazonaws.com/		processing		

Registry path	Version	Job types (image scope)		
sagemaker-spark-processing:<tag>				
411782140378.dkr.ecr.ap-3.0 northeast-1.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving', region='ap-northeast-1', version='2.4')
```

Registry path	Version	Job types (image scope)		
354813040037.dkr.ecr.ap-2.2 northeast-1.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		
354813040037.dkr.ecr.ap-2.4 northeast-1.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='ap-northeast-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ap-0.0 northeast-1.amazonaws.com/ sagemaker-tensorflow-eia:<tag>		eia	CPU	py2
520713654638.dkr.ecr.ap-0.0 northeast-1.amazonaws.com/ sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e-dr.10.0 northeast-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e-dr.10.1 northeast-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
520713654638.dkr.e-dr.10.0 northeast-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.10.0 northeast-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.10.0 northeast-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.10.0 northeast-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.10.0 northeast-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.10.0 northeast-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.10.0 northeast-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <del>4.0</del> -northeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>4.0</del> -northeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>5.0</del> -northeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>5.0</del> -northeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>6.0</del> -northeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>6.0</del> -northeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> -northeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> -northeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> -northeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> -northeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>9.0</del> -northeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>9.0</u> -northeast-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.e <u>10.0</u> -northeast-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>10.0</u> -northeast-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>10.0</u> -northeast-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>10.0</u> -northeast-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>10.0</u> -northeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>10.0</u> -northeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>10.0</u> -northeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>10.2</u> -northeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>10.3</u> -northeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.15</u> .4 northeast-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.15</u> .5 northeast-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.16</u> - northeast-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.16</u> - northeast-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.17</u> - northeast-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.17</u> - northeast-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.17</u> - northeast-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.18</u> - northeast-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.19</u> - northeast-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.19</u> - northeast-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.19</u> - northeast-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <del>0.20</del> . <sup>20</sup> -northeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.21</del> . <sup>21</sup> -northeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.22</del> . <sup>22</sup> -northeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.30</del> . <sup>30</sup> -northeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.31</del> . <sup>31</sup> -northeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.32</del> . <sup>32</sup> -northeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.41</del> . <sup>41</sup> -northeast-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.10-1</del> . <sup>10-1</sup> -northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <del>0.10</del> . <sup>10</sup> -northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <del>0.15</del> . <sup>15</sup> -northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <del>0.15</del> . <sup>15</sup> -northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.0.3</u> -northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.e <u>0.0.4</u> -northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0.0.5</u> -northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0.0.6</u> -northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.0.7</u> -northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.0.8</u> -northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.0.9</u> -northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.0.10</u> -northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.0.11</u> -northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.0.12</u> -northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e0.1p-northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e0.2p-northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.2p-northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.2p-northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.3p-northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.3p-northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.3p-northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.4p-northeast-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow',region='ap-northeast-1',version='1.0.0',image_scope='training',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.0p-northeast-1.amazonaws.com/sagemaker-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
rl-coach-container:coach-1.0.0-tf-<tag>				
520713654638.dkr.e0.10-northeast-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10.1-northeast-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11-northeast-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.0-northeast-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.1-northeast-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inferentia (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow',region='ap-northeast-1',version='1.15.0',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
941853720454.dkr.e0.15.0-northeast-1.amazonaws.com/sagemaker-neo-tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow', region='ap-northeast-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8.2-northeast-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.2-tf-<tag>	e0.8.2-	training	CPU, GPU	py36
462105765813.dkr.e0.8.5-northeast-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-tf-<tag>	e0.8.5-	training	CPU, GPU	py36
520713654638.dkr.e0.5.ap-northeast-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5-<tag>	e0.5.ap-	training	CPU, GPU	py3
520713654638.dkr.e0.5.ap-northeast-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3-<tag>	e0.5.ap-	training	CPU, GPU	py3
520713654638.dkr.e0.6.ap-northeast-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6-<tag>	e0.6.ap-	training	CPU, GPU	py3
520713654638.dkr.e0.6.ap-northeast-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5-<tag>	e0.6.ap-	training	CPU, GPU	py3

## VW (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='vw', region='ap-northeast-1', version='8.7.0', image_scope='training')
```

Registry path	Version	Job types (image scope)		
462105765813.dkr.ecr.ap-8.7.0-northeast-1.amazonaws.com/		training		

Registry path	Version	Job types (image scope)		
sagemaker-rl-vw-container:vw-8.7.0-<tag>				

### XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost',region='ap-northeast-1',version='1.2-1')
```

Registry path	Version	Job types (image scope)		
354813040037.dkr.ecr.ap-0.90-1 northeast-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
354813040037.dkr.ecr.ap-0.90-2 northeast-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
354813040037.dkr.ecr.ap-1.0-1 northeast-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
354813040037.dkr.ecr.ap-1.2-1 northeast-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
354813040037.dkr.ecr.ap-1.2-2 northeast-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
354813040037.dkr.ecr.ap-1.3-1 northeast-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
501404015308.dkr.ecr.ap-1 northeast-1.amazonaws.com/ xgboost:<tag>		inference, training		

### Docker Registry Paths and Example Code for Canada (Central) (ca-central-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

## Topics

- [BlazingText \(algorithm\) \(p. 1031\)](#)
- [Chainer \(DLC\) \(p. 1031\)](#)
- [Clarify \(algorithm\) \(p. 1031\)](#)
- [Data Wrangler \(algorithm\) \(p. 1032\)](#)
- [Debugger \(algorithm\) \(p. 1032\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 1032\)](#)
- [Factorization Machines \(algorithm\) \(p. 1033\)](#)
- [Hugging Face \(algorithm\) \(p. 1033\)](#)
- [IP Insights \(algorithm\) \(p. 1034\)](#)
- [Image classification \(algorithm\) \(p. 1034\)](#)
- [Inferentia MXNet \(DLC\) \(p. 1034\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 1034\)](#)
- [K-Means \(algorithm\) \(p. 1035\)](#)
- [KNN \(algorithm\) \(p. 1035\)](#)
- [LDA \(algorithm\) \(p. 1035\)](#)
- [Linear Learner \(algorithm\) \(p. 1036\)](#)
- [MXNet \(DLC\) \(p. 1036\)](#)
- [MXNet Coach \(DLC\) \(p. 1038\)](#)
- [Model Monitor \(algorithm\) \(p. 1039\)](#)
- [NTM \(algorithm\) \(p. 1039\)](#)
- [Neo Image Classification \(algorithm\) \(p. 1039\)](#)
- [Neo MXNet \(DLC\) \(p. 1040\)](#)
- [Neo PyTorch \(DLC\) \(p. 1040\)](#)
- [Neo Tensorflow \(DLC\) \(p. 1041\)](#)
- [Neo XGBoost \(algorithm\) \(p. 1041\)](#)
- [Object Detection \(algorithm\) \(p. 1041\)](#)
- [Object2Vec \(algorithm\) \(p. 1042\)](#)
- [PCA \(algorithm\) \(p. 1042\)](#)
- [PyTorch \(DLC\) \(p. 1042\)](#)
- [Random Cut Forest \(algorithm\) \(p. 1044\)](#)
- [Ray PyTorch \(DLC\) \(p. 1045\)](#)
- [Scikit-learn \(algorithm\) \(p. 1045\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 1045\)](#)
- [Seq2Seq \(algorithm\) \(p. 1046\)](#)
- [Spark \(algorithm\) \(p. 1046\)](#)
- [SparkML Serving \(algorithm\) \(p. 1046\)](#)
- [Tensorflow \(DLC\) \(p. 1047\)](#)
- [Tensorflow Coach \(DLC\) \(p. 1054\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 1055\)](#)
- [Tensorflow Ray \(DLC\) \(p. 1055\)](#)
- [VW \(algorithm\) \(p. 1056\)](#)
- [XGBoost \(algorithm\) \(p. 1056\)](#)

## BlazingText (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext',region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1-central-1.amazonaws.com/blazingtext:<tag>		inference, training		

## Chainer (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer',region='ca-central-1',version='5.0.0',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ca-1-central-1.amazonaws.com/sagemaker-chainer:<tag>	4.0.0	inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ca-1-central-1.amazonaws.com/sagemaker-chainer:<tag>	4.1.0	inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ca-1-central-1.amazonaws.com/sagemaker-chainer:<tag>	5.0.0	inference, training	CPU, GPU	py2, py3

## Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify',region='ca-central-1',version='1.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
675030665977.dkr.ecr.ca-1-central-1.amazonaws.com/		processing		

Registry path	Version	Job types (image scope)		
sagemaker-clarify-processing:<tag>				

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler', region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
557239378090.dkr.ecr.ca-1.x central-1.amazonaws.com/ sagemaker-data-wrangler-container:<tag>		processing		

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger', region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
519511493484.dkr.ecr.ca-latest central-1.amazonaws.com/ sagemaker-debugger-rules:<tag>		debugger		

### DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar', region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1 central-1.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
forecasting-deepar:<tag>				

### Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines', region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1-central-1.amazonaws.com/factorization-machines:<tag>		inference, training		

### Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='ca-central-1', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.ca-4.4.2-central-1.amazonaws.com/huggingface-pytorch-training:<tag>		training		
763104351884.dkr.ecr.ca-4.5.0-central-1.amazonaws.com/huggingface-pytorch-training:<tag>		training		
763104351884.dkr.ecr.ca-4.4.2-central-1.amazonaws.com/huggingface-tensorflow-training:<tag>		training		
763104351884.dkr.ecr.ca-4.5.0-central-1.amazonaws.com/huggingface-tensorflow-training:<tag>		training		

## IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights',region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1 central-1.amazonaws.com/ ipinsights:<tag>		inference, training		

## Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification',region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1 central-1.amazonaws.com/ image- classification:<tag>		inference, training		

## Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='ca-  
central-1',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
464438896020.dkr.ecr.ca-1 central-1.amazonaws.com/ sagemaker-neo- mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='ca-  
central-1',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
464438896020.dkr.ecr.ca-1-central-1.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

### K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans', region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1-central-1.amazonaws.com/kmeans:<tag>		inference, training		

### KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn', region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1-central-1.amazonaws.com/knn:<tag>		inference, training		

### LDA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='lda', region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1-central-1.amazonaws.com/lda:<tag>		inference, training		

## Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1 central-1.amazonaws.com/ linear-learner:<tag>		inference, training		

## MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='ca-  
central-1', version='1.4.1', py_version='py3', image_scope='inference',  
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>dr.30</u> central-1.amazonaws.com/ sagemaker-mxnet- eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.e <u>dr.40</u> central-1.amazonaws.com/ sagemaker-mxnet- serving-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.e <u>dr.40</u> central-1.amazonaws.com/ sagemaker-mxnet- serving:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>dr.41</u> central-1.amazonaws.com/ sagemaker-mxnet- serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e <u>0.12.1</u> central-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.12.1</u> central-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <del>00</del> central-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <del>00</del> central-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <del>10</del> central-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <del>10</del> central-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <del>20</del> central-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <del>20</del> central-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <del>30</del> central-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <del>30</del> central-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <del>40</del> central-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <del>40</del> central-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <del>40</del> central-1.amazonaws.com/ mxnet-inference- eia:<tag>		eia	CPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr. <del>5a1</del> central-1.amazonaws.com/ mxnet-inference- eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.ecr. <del>7a0</del> central-1.amazonaws.com/ mxnet-inference- eia:<tag>		eia	CPU	py3
763104351884.dkr.ecr. <del>4a1</del> central-1.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.ecr. <del>6a0</del> central-1.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.ecr. <del>7a0</del> central-1.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.ecr. <del>8a0</del> central-1.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py37
763104351884.dkr.ecr. <del>4a1</del> central-1.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.ecr. <del>6a0</del> central-1.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.ecr. <del>7a0</del> central-1.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.ecr. <del>8a0</del> central-1.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py37

## MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='coach-mxnet',region='ca-central-1',version='0.11',py_version='py3',image_scope='training',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.t1-central-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.t1.0-central-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor',region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
536280801234.dkr.ecr.ca-central-1.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm',region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1-central-1.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='image-classification-neo',region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
464438896020.dkr.ecr.ca-latest central-1.amazonaws.com/ image-classification- neo:<tag>		inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet',region='ca-
central-1',version='1.8',py_version='py3',image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
464438896020.dkr.ecr.ca- central-1.amazonaws.com/ sagemaker- inference- mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch',region='ca-
central-1',version='1.6',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
464438896020.dkr.ecr.ca- central-1.amazonaws.com/ sagemaker- inference- pytorch:<tag>		inference	CPU, GPU	py3
464438896020.dkr.ecr.ca- central-1.amazonaws.com/ sagemaker- inference- pytorch:<tag>		inference	CPU, GPU	py3
464438896020.dkr.ecr.ca- central-1.amazonaws.com/		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-inference-pytorch:<tag>				

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='ca-central-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
464438896020.dkr.ecr.ca-central-1.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

### Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
464438896020.dkr.ecr.ca-latestcentral-1.amazonaws.com/xgboost-neo:<tag>		inference		

### Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1central-1.amazonaws.com/object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec',region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1 central-1.amazonaws.com/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca',region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1 central-1.amazonaws.com/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch',region='ca-
central-1',version='1.8.0',py_version='py3',image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.40 central-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.40 central-1.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e0.40 central-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>dr.0</u> central-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>dr.1</u> central-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>dr.10</u> central-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>dr.1</u> central-1.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.e <u>dr.2</u> central-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.e <u>dr.3</u> central-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.e <u>dr.4</u> central-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.e <u>dr.5</u> central-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.e <u>dr.6</u> central-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.e <u>dr.7</u> central-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.e <u>dr.8</u> central-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr.81-central-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr.20-central-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.71-central-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.40-central-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.50-central-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr.60-central-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr.70-central-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr.80-central-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr.81-central-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

## Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1 central-1.amazonaws.com/ randomcutforest:<tag>		inference, training		

## Ray PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-pytorch', region='ca-
central-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8.5- central-1.amazonaws.com/ sagemaker-rl-ray- container:ray-0.8.5- torch-<tag>	0.8.5	training	CPU, GPU	py36

## Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn', region='ca-
central-1', version='0.23-1', image_scope='inference')
```

Registry path	Version	Job types (image scope)		
341280168497.dkr.ecr.ca-0.20.0 central-1.amazonaws.com/ sagemaker-scikit- learn:<tag>		inference, training		
341280168497.dkr.ecr.ca-0.23-1 central-1.amazonaws.com/ sagemaker-scikit- learn:<tag>		inference, training		

## Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='semantic-segmentation',region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1central-1.amazonaws.com/semantic-segmentation:<tag>		inference, training		

## Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='ca-central-1')
```

Registry path	Version	Job types (image scope)		
469771592824.dkr.ecr.ca-1central-1.amazonaws.com/seq2seq:<tag>		inference, training		

## Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='ca-central-1',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
446299261295.dkr.ecr.ca-2.4central-1.amazonaws.com/sagemaker-spark-processing:<tag>		processing		
446299261295.dkr.ecr.ca-3.0central-1.amazonaws.com/sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving',region='ca-central-1',version='2.4')
```

Registry path	Version	Job types (image scope)		
341280168497.dkr.ecr.ca-2.2 central-1.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		
341280168497.dkr.ecr.ca-2.4 central-1.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		

### Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='ca-  
central-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.ca-0 central-1.amazonaws.com/ sagemaker- tensorflow- eia:<tag>		eia	CPU	py2
520713654638.dkr.ecr.ca-0 central-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ca-0 central-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.ecr.ca-1 central-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr.ca-0 central-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.ecr.ca-0 central-1.amazonaws.com/		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-tensorflow-serving-eia:<tag>				
520713654638.dkr.e-dr.t <small>a</small> :0 central-1.amazonaws.com/ sagemaker-tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.t <small>a</small> :0 central-1.amazonaws.com/ sagemaker-tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.t <small>a</small> :0 central-1.amazonaws.com/ sagemaker-tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.t <small>0</small> :0 central-1.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.t <small>0</small> :0 central-1.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.e-dr.t <small>1</small> :0 central-1.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.t <small>1</small> :0 central-1.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.e-dr.t <small>2</small> :0 central-1.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.t <small>2</small> :0 central-1.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.e-dr.t <small>3</small> :0 central-1.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <del>6.0</del> central-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> central-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> central-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> central-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> central-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>9.0</del> central-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>9.0</del> central-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <del>14.0</del> central-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <del>15.0</del> central-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <del>16.0</del> central-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.70</u> .0 central-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>0.70</u> .0 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.70</u> .0 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.70</u> .0 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.70</u> .2 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.70</u> .3 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.70</u> .4 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.70</u> .5 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.00</u> .0 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.00</u> .1 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.00</u> .2 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.03</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.04</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.10</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.11</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.12</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.13</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.14</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.15</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.16</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.17</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.18</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.19</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.20</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.21</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.22</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.30</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.31</u> -central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.1<u>2</u></u> central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.1<u>4</u>-1</u> central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.1<u>5</u>-1</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.1<u>5</u>-0</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.1<u>5</u>-0</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.1<u>5</u>-2</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.e <u>0.1<u>5</u>-3</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.e <u>0.1<u>5</u>-4</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0.1<u>5</u>-5</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0.0<u>0</u>-0</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.0<u>1</u>-1</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.02</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.03</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.04</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.10</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.11</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.12</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.13</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.20</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.21</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.22</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.30</u> central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e0.31-central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.32-central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.41-central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow', region='ca-central-1', version='1.0.0', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.00-central-1.amazonaws.com/sagemaker-rl-coach-container:coach-1.0.0-tf-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10-central-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10-1-central-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11-central-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11-0-central-1.amazonaws.com/		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-rl-tensorflow:coach0.11.0-<tag>				
520713654638.dkr.e0.1a-1 central-1.amazonaws.com/ sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inference (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow', region='ca-central-1', version='1.15.0', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
464438896020.dkr.e0.1a-0 central-1.amazonaws.com/ sagemaker-neo-tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow', region='ca-central-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8a-2 central-1.amazonaws.com/ sagemaker-rl-ray-container:ray-0.8.2-tf-<tag>		training	CPU, GPU	py36
462105765813.dkr.e0.8a-5 central-1.amazonaws.com/ sagemaker-rl-ray-container:ray-0.8.5-tf-<tag>		training	CPU, GPU	py36
520713654638.dkr.e0.5a- central-1.amazonaws.com/		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-rl-tensorflow:ray0.5-<tag>				
520713654638.dkr.ecr.ca-central-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
520713654638.dkr.ecr.ca-central-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6-<tag>		training	CPU, GPU	py3
520713654638.dkr.ecr.ca-central-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5-<tag>		training	CPU, GPU	py3

## VW (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='vw', region='ca-central-1', version='8.7.0', image_scope='training')
```

Registry path	Version	Job types (image scope)		
462105765813.dkr.ecr.ca-8.7.0 central-1.amazonaws.com/sagemaker-rl-vw-container:vw-8.7.0-<tag>		training		

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost', region='ca-central-1', version='1.2-1')
```

Registry path	Version	Job types (image scope)		
341280168497.dkr.ecr.ca-0.90-1 central-1.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
sagemaker-xgboost:<tag>				
341280168497.dkr.ecr.ca-0.90-2 central-1.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
341280168497.dkr.ecr.ca-1.0-1 central-1.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
341280168497.dkr.ecr.ca-1.2-1 central-1.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
341280168497.dkr.ecr.ca-1.2-2 central-1.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
341280168497.dkr.ecr.ca-1.3-1 central-1.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
469771592824.dkr.ecr.ca-1 central-1.amazonaws.com/ xgboost:<tag>		inference, training		

## Docker Registry Paths and Example Code for China (Beijing) (cn-north-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

### Topics

- [BlazingText \(algorithm\) \(p. 1058\)](#)
- [Chainer \(DLC\) \(p. 1058\)](#)
- [Clarify \(algorithm\) \(p. 1059\)](#)
- [Data Wrangler \(algorithm\) \(p. 1059\)](#)
- [Debugger \(algorithm\) \(p. 1060\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 1060\)](#)
- [Factorization Machines \(algorithm\) \(p. 1060\)](#)
- [Hugging Face \(algorithm\) \(p. 1060\)](#)
- [IP Insights \(algorithm\) \(p. 1061\)](#)
- [Image classification \(algorithm\) \(p. 1061\)](#)
- [Inferentia MXNet \(DLC\) \(p. 1062\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 1062\)](#)
- [K-Means \(algorithm\) \(p. 1062\)](#)

- [KNN \(algorithm\) \(p. 1062\)](#)
- [Linear Learner \(algorithm\) \(p. 1063\)](#)
- [MXNet \(DLC\) \(p. 1063\)](#)
- [MXNet Coach \(DLC\) \(p. 1066\)](#)
- [Model Monitor \(algorithm\) \(p. 1066\)](#)
- [NTM \(algorithm\) \(p. 1066\)](#)
- [Neo Image Classification \(algorithm\) \(p. 1067\)](#)
- [Neo MXNet \(DLC\) \(p. 1067\)](#)
- [Neo PyTorch \(DLC\) \(p. 1067\)](#)
- [Neo Tensorflow \(DLC\) \(p. 1068\)](#)
- [Neo XGBoost \(algorithm\) \(p. 1068\)](#)
- [Object Detection \(algorithm\) \(p. 1069\)](#)
- [Object2Vec \(algorithm\) \(p. 1069\)](#)
- [PCA \(algorithm\) \(p. 1069\)](#)
- [PyTorch \(DLC\) \(p. 1069\)](#)
- [Random Cut Forest \(algorithm\) \(p. 1072\)](#)
- [Scikit-learn \(algorithm\) \(p. 1072\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 1072\)](#)
- [Seq2Seq \(algorithm\) \(p. 1073\)](#)
- [Spark \(algorithm\) \(p. 1073\)](#)
- [SparkML Serving \(algorithm\) \(p. 1073\)](#)
- [Tensorflow \(DLC\) \(p. 1074\)](#)
- [Tensorflow Coach \(DLC\) \(p. 1081\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 1082\)](#)
- [Tensorflow Ray \(DLC\) \(p. 1082\)](#)
- [XGBoost \(algorithm\) \(p. 1083\)](#)

### BlazingText (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext',region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ blazingtext:<tag>		inference, training		

### Chainer (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='chainer', region='cn-north-1', version='5.0.0', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
422961961927.dkr.ecr.cn-1.0-north-1.amazonaws.com.cn/sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3
422961961927.dkr.ecr.cn-1.0-north-1.amazonaws.com.cn/sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3
422961961927.dkr.ecr.cn-1.0-north-1.amazonaws.com.cn/sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3

### Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify', region='cn-north-1', version='1.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
122526803553.dkr.ecr.cn-1.0-north-1.amazonaws.com.cn/sagemaker-clarify-processing:<tag>		processing		

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler', region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
245909111842.dkr.ecr.cn-1.x-north-1.amazonaws.com.cn/sagemaker-data-wrangler-container:<tag>		processing		

## Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger', region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
618459771430.dkr.ecr.cn-latest north-1.amazonaws.com.cn/ sagemaker-debugger- rules:<tag>		debugger		

## DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar', region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ forecasting- deepar:<tag>		inference, training		

## Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines', region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ factorization- machines:<tag>		inference, training		

## Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='cn-  
north-1', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
727897471807.dkr.ecr.cn-4.4.2 north-1.amazonaws.com.cn/ huggingface-pytorch- training:<tag>		training		
727897471807.dkr.ecr.cn-4.5.0 north-1.amazonaws.com.cn/ huggingface-pytorch- training:<tag>		training		
727897471807.dkr.ecr.cn-4.4.2 north-1.amazonaws.com.cn/ huggingface- tensorflow- training:<tag>		training		
727897471807.dkr.ecr.cn-4.5.0 north-1.amazonaws.com.cn/ huggingface- tensorflow- training:<tag>		training		

## IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights', region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ ipinsights:<tag>		inference, training		

## Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification', region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ image- classification:<tag>		inference, training		

## Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='cn-north-1',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
472730292857.dkr.ecr.cn-north-1.amazonaws.com.cn/sagemaker-neo-mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='cn-north-1',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
472730292857.dkr.ecr.cn-north-1.amazonaws.com.cn/sagemaker-neo-pytorch:<tag>		inference	inf	py3

## K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans',region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-north-1.amazonaws.com.cn/kmeans:<tag>		inference, training		

## KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn',region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ knn:<tag>		inference, training		

### Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ linear-learner:<tag>		inference, training		

### MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='cn-north-1', version='1.4.1', py_version='py3', image_scope='inference',
    instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
422961961927.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3
422961961927.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
422961961927.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
422961961927.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
422961961927.dkr.e0.10.1 north-1.amazonaws.com.cn/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
422961961927.dkr.e0.10.1 north-1.amazonaws.com.cn/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
422961961927.dkr.e0.00- north-1.amazonaws.com.cn/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
422961961927.dkr.e0.00- north-1.amazonaws.com.cn/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
422961961927.dkr.e0.10- north-1.amazonaws.com.cn/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
422961961927.dkr.e0.10- north-1.amazonaws.com.cn/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
422961961927.dkr.e0.21- north-1.amazonaws.com.cn/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
422961961927.dkr.e0.21- north-1.amazonaws.com.cn/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
422961961927.dkr.e0.30- north-1.amazonaws.com.cn/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
422961961927.dkr.e0.30- north-1.amazonaws.com.cn/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
422961961927.dkr.e0.40- north-1.amazonaws.com.cn/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
422961961927.dkr.edr. <b>4.1</b> -north-1.amazonaws.com.cn/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
727897471807.dkr.edr. <b>4.1</b> -north-1.amazonaws.com.cn/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
727897471807.dkr.edr. <b>5.1</b> -north-1.amazonaws.com.cn/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
727897471807.dkr.edr. <b>7.0</b> -north-1.amazonaws.com.cn/mxnet-inference-eia:<tag>		eia	CPU	py3
727897471807.dkr.edr. <b>4.1</b> -north-1.amazonaws.com.cn/mxnet-inference:<tag>		inference	CPU, GPU	py3
727897471807.dkr.edr. <b>6.0</b> -north-1.amazonaws.com.cn/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
727897471807.dkr.edr. <b>7.0</b> -north-1.amazonaws.com.cn/mxnet-inference:<tag>		inference	CPU, GPU	py3
727897471807.dkr.edr. <b>8.0</b> -north-1.amazonaws.com.cn/mxnet-inference:<tag>		inference	CPU, GPU	py37
727897471807.dkr.edr. <b>4.1</b> -north-1.amazonaws.com.cn/mxnet-training:<tag>		training	CPU, GPU	py3
727897471807.dkr.edr. <b>6.0</b> -north-1.amazonaws.com.cn/mxnet-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.edr. <b>7.0</b> -north-1.amazonaws.com.cn/mxnet-training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
727897471807.dkr.ecr.cn-north-1.amazonaws.com.cn/mxnet-training:<tag>		training	CPU, GPU	py37

### MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='cn-north-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
422961961927.dkr.ecr.cn-north-1.amazonaws.com.cn/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
422961961927.dkr.ecr.cn-north-1.amazonaws.com.cn/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor', region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
453000072557.dkr.ecr.cn-north-1.amazonaws.com.cn/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='ntm', region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo', region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
472730292857.dkr.ecr.cn-latest north-1.amazonaws.com.cn/ image-classification- neo:<tag>		inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet', region='cn-north-1', version='1.8', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
472730292857.dkr.ecr.cn- north-1.amazonaws.com.cn/ sagemaker- inference- mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='cn-north-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
472730292857.dkr.ecr.cn-north-1.amazonaws.com.cn/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
472730292857.dkr.ecr.cn-north-1.amazonaws.com.cn/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
472730292857.dkr.ecr.cn-north-1.amazonaws.com.cn/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='cn-north-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
472730292857.dkr.ecr.cn-north-1.amazonaws.com.cn/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

### Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
472730292857.dkr.ecr.cn-latest		inference		
north-1.amazonaws.com.cn/xgboost-neo:<tag>				

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection',region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec',region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca',region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch',region='cn-
north-1',version='1.8.0',py_version='py3',image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
422961961927.dkr.e0.410-north-1.amazonaws.com.cn/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
422961961927.dkr.e0.410-north-1.amazonaws.com.cn/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
422961961927.dkr.edr.010-north-1.amazonaws.com.cn/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
422961961927.dkr.edr.010-north-1.amazonaws.com.cn/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
422961961927.dkr.edr.110-north-1.amazonaws.com.cn/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
422961961927.dkr.edr.110-north-1.amazonaws.com.cn/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.edr.311-north-1.amazonaws.com.cn/pytorch-inference-eia:<tag>		eia	CPU	py3
727897471807.dkr.edr.210-north-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
727897471807.dkr.edr.311-north-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
727897471807.dkr.edr.410-north-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py3
727897471807.dkr.edr.510-north-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
727897471807.dkr.edr.610-north-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
727897471807.dkr.edr.711-north-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
727897471807.dkr.edr.810-north-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
727897471807.dkr.edr.811-north-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
727897471807.dkr.edr.210-north-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.edr.311-north-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.edr.410-north-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.edr.510-north-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py3
727897471807.dkr.edr.610-north-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py3, py36
727897471807.dkr.edr.711-north-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py3, py36
727897471807.dkr.edr.810-north-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py3, py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
727897471807.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ pytorch-training:<tag>		training	CPU, GPU	py3, py36

### Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest',region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ randomcutforest:<tag>		inference, training		

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn',region='cn-north-1',version='0.23-1',image_scope='inference')
```

Registry path	Version	Job types (image scope)		
450853457545.dkr.ecr.cn-0.20.0 north-1.amazonaws.com.cn/ sagemaker-scikit-learn:<tag>		inference, training		
450853457545.dkr.ecr.cn-0.23-1 north-1.amazonaws.com.cn/ sagemaker-scikit-learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ semantic-segmentation:<tag>		inference, training		

## Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='cn-north-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ seq2seq:<tag>		inference, training		

## Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='cn-north-1',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
671472414489.dkr.ecr.cn-2.4 north-1.amazonaws.com.cn/ sagemaker-spark-processing:<tag>		processing		
671472414489.dkr.ecr.cn-3.0 north-1.amazonaws.com.cn/ sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving',region='cn-north-1',version='2.4')
```

Registry path	Version	Job types (image scope)		
450853457545.dkr.ecr.cn-2.2 north-1.amazonaws.com.cn/ sagemaker-sparkml- serving:<tag>		inference		
450853457545.dkr.ecr.cn-2.4 north-1.amazonaws.com.cn/ sagemaker-sparkml- serving:<tag>		inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='cn-north-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
422961961927.dkr.ecr.t0:0 north-1.amazonaws.com.cn/ sagemaker- tensorflow- eia:<tag>		eia	CPU	py2
422961961927.dkr.ecr.t1:0 north-1.amazonaws.com.cn/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
422961961927.dkr.ecr.t2:0 north-1.amazonaws.com.cn/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
422961961927.dkr.ecr.t3:1 north-1.amazonaws.com.cn/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
422961961927.dkr.ecr.t3:0 north-1.amazonaws.com.cn/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
422961961927.dkr.ecr.t0:0 north-1.amazonaws.com.cn/		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-tensorflow-serving-eia:<tag>				
422961961927.dkr.e-dr.t0:0 north-1.amazonaws.com.cn/ sagemaker-tensorflow- serving-eia:<tag>		eia	CPU	-
422961961927.dkr.e-dr.t0:0 north-1.amazonaws.com.cn/ sagemaker-tensorflow- serving:<tag>		inference	CPU, GPU	-
422961961927.dkr.e-dr.t0:0 north-1.amazonaws.com.cn/ sagemaker-tensorflow- serving:<tag>		inference	CPU, GPU	-
422961961927.dkr.e-dr.t0:0 north-1.amazonaws.com.cn/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
422961961927.dkr.e-dr.t0:0 north-1.amazonaws.com.cn/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
422961961927.dkr.e-dr.t1- north-1.amazonaws.com.cn/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
422961961927.dkr.e-dr.t1- north-1.amazonaws.com.cn/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
422961961927.dkr.e-dr.t0- north-1.amazonaws.com.cn/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
422961961927.dkr.e-dr.t0- north-1.amazonaws.com.cn/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
422961961927.dkr.e-dr.t0- north-1.amazonaws.com.cn/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
422961961927.dkr.edr.610-north-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
422961961927.dkr.edr.710-north-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
422961961927.dkr.edr.710-north-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
422961961927.dkr.edr.810-north-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
422961961927.dkr.edr.810-north-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
422961961927.dkr.edr.910-north-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
422961961927.dkr.edr.910-north-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
727897471807.dkr.edr.14.0-north-1.amazonaws.com.cn/tensorflow-inference-eia:<tag>		eia	CPU	-
727897471807.dkr.edr.15.0-north-1.amazonaws.com.cn/tensorflow-inference-eia:<tag>		eia	CPU	-
727897471807.dkr.edr.010-north-1.amazonaws.com.cn/tensorflow-inference-eia:<tag>		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
727897471807.dkr.e <u>0.3.0</u> -north-1.amazonaws.com.cn/tensorflow-inference-eia:<tag>		eia	CPU	-
727897471807.dkr.e <u>0.3.0</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.3.0</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.3.0</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.3.2</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.3.3</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.3.4</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.3.5</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.0.0</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.0.1</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.0.2</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
727897471807.dkr.e <u>0.0.3</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.0.4</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.1.0</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.1.1</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.1.2</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.1.3</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.2.0</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.2.1</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.2.2</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.3.0</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.3.1</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
727897471807.dkr.e <u>0.3.2</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.4.1</u> -north-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.1.1</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py3
727897471807.dkr.e <u>0.1.0</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.e <u>0.1.0</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.e <u>0.1.2</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37
727897471807.dkr.e <u>0.1.3</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37
727897471807.dkr.e <u>0.1.4</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
727897471807.dkr.e <u>0.1.5</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
727897471807.dkr.e <u>0.0.0</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.e <u>0.0.1</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
727897471807.dkr.e <u>0.0.2</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.e <u>0.0.3</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py3
727897471807.dkr.e <u>0.0.4</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py3
727897471807.dkr.e <u>0.1.0</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.e <u>0.1.1</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.e <u>0.1.2</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py3
727897471807.dkr.e <u>0.1.3</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py3
727897471807.dkr.e <u>0.2.0</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py37
727897471807.dkr.e <u>0.2.1</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py37
727897471807.dkr.e <u>0.2.2</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py37
727897471807.dkr.e <u>0.3.0</u> -north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
727897471807.dkr.e0.3.1-north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py37
727897471807.dkr.e0.3.2-north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py37
727897471807.dkr.e0.4.1-north-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow', region='cn-north-1', version='1.0.0', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
422961961927.dkr.e0.10-north-1.amazonaws.com.cn/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
422961961927.dkr.e0.10.1-north-1.amazonaws.com.cn/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
422961961927.dkr.e0.11-north-1.amazonaws.com.cn/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
422961961927.dkr.e0.11.0-north-1.amazonaws.com.cn/sagemaker-rl-tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
422961961927.dkr.e0.11.1-north-1.amazonaws.com.cn/sagemaker-rl-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow:coach0.11.1- <i>&lt;tag&gt;</i>				

## Tensorflow Inference (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow',region='cn-north-1',version='1.15.0',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
472730292857.dkr.e0r.t5.0 north-1.amazonaws.com.cn/ sagemaker-neo-tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow',region='cn-north-1',version='0.8.5',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
422961961927.dkr.e0r.t5.0 north-1.amazonaws.com.cn/ sagemaker-rl-tensorflow:ray0.5-<tag>		training	CPU, GPU	py3
422961961927.dkr.e0r.t5.3 north-1.amazonaws.com.cn/ sagemaker-rl-tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
422961961927.dkr.e0r.t5.0 north-1.amazonaws.com.cn/ sagemaker-rl-tensorflow:ray0.6-<tag>		training	CPU, GPU	py3
422961961927.dkr.e0r.t5.0 north-1.amazonaws.com.cn/ sagemaker-rl-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow:ray0.6.5-<tag>				

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost', region='cn-north-1', version='1.2-1')
```

Registry path	Version	Job types (image scope)		
390948362332.dkr.ecr.cn-1 north-1.amazonaws.com.cn/ xgboost:<tag>		inference, training		
450853457545.dkr.ecr.cn-0.90-1 north-1.amazonaws.com.cn/ sagemaker- xgboost:<tag>		inference, training		
450853457545.dkr.ecr.cn-0.90-2 north-1.amazonaws.com.cn/ sagemaker- xgboost:<tag>		inference, training		
450853457545.dkr.ecr.cn-1.0-1 north-1.amazonaws.com.cn/ sagemaker- xgboost:<tag>		inference, training		
450853457545.dkr.ecr.cn-1.2-1 north-1.amazonaws.com.cn/ sagemaker- xgboost:<tag>		inference, training		
450853457545.dkr.ecr.cn-1.2-2 north-1.amazonaws.com.cn/ sagemaker- xgboost:<tag>		inference, training		
450853457545.dkr.ecr.cn-1.3-1 north-1.amazonaws.com.cn/ sagemaker- xgboost:<tag>		inference, training		

## Docker Registry Paths and Example Code for China (Ningxia) (cn-northwest-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

### Topics

- [BlazingText \(algorithm\) \(p. 1084\)](#)
- [Chainer \(DLC\) \(p. 1085\)](#)
- [Clarify \(algorithm\) \(p. 1085\)](#)
- [Data Wrangler \(algorithm\) \(p. 1086\)](#)
- [Debugger \(algorithm\) \(p. 1086\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 1086\)](#)
- [Factorization Machines \(algorithm\) \(p. 1086\)](#)
- [Hugging Face \(algorithm\) \(p. 1087\)](#)
- [IP Insights \(algorithm\) \(p. 1087\)](#)
- [Image classification \(algorithm\) \(p. 1088\)](#)
- [Inferentia MXNet \(DLC\) \(p. 1088\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 1088\)](#)
- [K-Means \(algorithm\) \(p. 1088\)](#)
- [KNN \(algorithm\) \(p. 1089\)](#)
- [Linear Learner \(algorithm\) \(p. 1089\)](#)
- [MXNet \(DLC\) \(p. 1089\)](#)
- [MXNet Coach \(DLC\) \(p. 1092\)](#)
- [Model Monitor \(algorithm\) \(p. 1092\)](#)
- [NTM \(algorithm\) \(p. 1093\)](#)
- [Neo Image Classification \(algorithm\) \(p. 1093\)](#)
- [Neo MXNet \(DLC\) \(p. 1093\)](#)
- [Neo PyTorch \(DLC\) \(p. 1094\)](#)
- [Neo Tensorflow \(DLC\) \(p. 1094\)](#)
- [Neo XGBoost \(algorithm\) \(p. 1095\)](#)
- [Object Detection \(algorithm\) \(p. 1095\)](#)
- [Object2Vec \(algorithm\) \(p. 1095\)](#)
- [PCA \(algorithm\) \(p. 1095\)](#)
- [PyTorch \(DLC\) \(p. 1096\)](#)
- [Random Cut Forest \(algorithm\) \(p. 1098\)](#)
- [Scikit-learn \(algorithm\) \(p. 1098\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 1099\)](#)
- [Seq2Seq \(algorithm\) \(p. 1099\)](#)
- [Spark \(algorithm\) \(p. 1099\)](#)
- [SparkML Serving \(algorithm\) \(p. 1100\)](#)
- [Tensorflow \(DLC\) \(p. 1100\)](#)
- [Tensorflow Coach \(DLC\) \(p. 1107\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 1108\)](#)
- [Tensorflow Ray \(DLC\) \(p. 1108\)](#)
- [XGBoost \(algorithm\) \(p. 1109\)](#)

### [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext', region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ blazingtext:<tag>		inference, training		

## Chainer (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer', region='cn-northwest-1', version='5.0.0', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
423003514399.dkr.ecr.cn-0 northwest-1.amazonaws.com.cn/ sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3
423003514399.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3
423003514399.dkr.ecr.cn-2 northwest-1.amazonaws.com.cn/ sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3

## Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify', region='cn-northwest-1', version='1.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
122578899357.dkr.ecr.cn-1.0 northwest-1.amazonaws.com.cn/ sagemaker-clarify-processing:<tag>		processing		

## Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler',region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
249157047649.dkr.ecr.cn-1.x northwest-1.amazonaws.com.cn/ sagemaker- data-wrangler- container:<tag>		processing		

## Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger',region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
658757709296.dkr.ecr.cn-latest northwest-1.amazonaws.com.cn/ sagemaker-debugger- rules:<tag>		debugger		

## DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar',region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ forecasting- deepar:<tag>		inference, training		

## Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines',region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ factorization- machines:<tag>		inference, training		

## Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='cn-northwest-1', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
727897471807.dkr.ecr.cn-4.4.2 northwest-1.amazonaws.com.cn/ huggingface-pytorch- training:<tag>		training		
727897471807.dkr.ecr.cn-4.5.0 northwest-1.amazonaws.com.cn/ huggingface-pytorch- training:<tag>		training		
727897471807.dkr.ecr.cn-4.4.2 northwest-1.amazonaws.com.cn/ huggingface- tensorflow- training:<tag>		training		
727897471807.dkr.ecr.cn-4.5.0 northwest-1.amazonaws.com.cn/ huggingface- tensorflow- training:<tag>		training		

## IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights', region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ ipinsights:<tag>		inference, training		

## Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification', region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ image- classification:<tag>		inference, training		

## Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet', region='cn-  
northwest-1', version='1.5.1', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
474822919863.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ sagemaker-neo- mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch', region='cn-  
northwest-1', version='1.5.1', py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
474822919863.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ sagemaker-neo- pytorch:<tag>		inference	inf	py3

## K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='kmeans', region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ kmeans:<tag>		inference, training		

### KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn', region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ knn:<tag>		inference, training		

### Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ linear-learner:<tag>		inference, training		

### MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='cn-northwest-1', version='1.4.1', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
423003514399.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/		eia	CPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-mxnet-eia:<tag>				
423003514399.dkr.edr.410-northwest-1.amazonaws.com.cn/sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
423003514399.dkr.edr.410-northwest-1.amazonaws.com.cn/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
423003514399.dkr.edr.411-northwest-1.amazonaws.com.cn/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2
423003514399.dkr.e0.10.1-northwest-1.amazonaws.com.cn/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
423003514399.dkr.e0.10.1-northwest-1.amazonaws.com.cn/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
423003514399.dkr.edr.010-northwest-1.amazonaws.com.cn/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
423003514399.dkr.edr.010-northwest-1.amazonaws.com.cn/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
423003514399.dkr.edr.110-northwest-1.amazonaws.com.cn/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
423003514399.dkr.edr.110-northwest-1.amazonaws.com.cn/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
423003514399.dkr.edr.211-northwest-1.amazonaws.com.cn/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
423003514399.dkr.edr.21-northwest-1.amazonaws.com.cn/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
423003514399.dkr.edr.30-northwest-1.amazonaws.com.cn/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
423003514399.dkr.edr.30-northwest-1.amazonaws.com.cn/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
423003514399.dkr.edr.40-northwest-1.amazonaws.com.cn/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
423003514399.dkr.edr.41-northwest-1.amazonaws.com.cn/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
727897471807.dkr.edr.41-northwest-1.amazonaws.com.cn/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
727897471807.dkr.edr.51-northwest-1.amazonaws.com.cn/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
727897471807.dkr.edr.70-northwest-1.amazonaws.com.cn/mxnet-inference-eia:<tag>		eia	CPU	py3
727897471807.dkr.edr.41-northwest-1.amazonaws.com.cn/mxnet-inference:<tag>		inference	CPU, GPU	py3
727897471807.dkr.edr.60-northwest-1.amazonaws.com.cn/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
727897471807.dkr.edr.70-northwest-1.amazonaws.com.cn/mxnet-inference:<tag>		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
727897471807.dkr.e0.810-northwest-1.amazonaws.com.cn/mxnet-inference:<tag>		inference	CPU, GPU	py37
727897471807.dkr.e0.411-northwest-1.amazonaws.com.cn/mxnet-training:<tag>		training	CPU, GPU	py3
727897471807.dkr.e0.610-northwest-1.amazonaws.com.cn/mxnet-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.e0.710-northwest-1.amazonaws.com.cn/mxnet-training:<tag>		training	CPU, GPU	py3
727897471807.dkr.e0.810-northwest-1.amazonaws.com.cn/mxnet-training:<tag>		training	CPU, GPU	py37

## MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='cn-northwest-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
423003514399.dkr.e0.110-northwest-1.amazonaws.com.cn/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
423003514399.dkr.e0.110.0-northwest-1.amazonaws.com.cn/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

## Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='model-monitor',region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
453252182341.dkr.ecr.cn-northwest-1.amazonaws.com.cn/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm',region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1northwest-1.amazonaws.com.cn/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo',region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
474822919863.dkr.ecr.cn-latestnorthwest-1.amazonaws.com.cn/image-classification-neo:<tag>		inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet',region='cn-northwest-1',version='1.8',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
474822919863.dkr.ecr.cn-northwest-1.amazonaws.com.cn/sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='cn-northwest-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
474822919863.dkr.ecr.cn-northwest-1.amazonaws.com.cn/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
474822919863.dkr.ecr.cn-northwest-1.amazonaws.com.cn/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
474822919863.dkr.ecr.cn-northwest-1.amazonaws.com.cn/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='cn-northwest-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
474822919863.dkr.ecr.cn-northwest-1.amazonaws.com.cn/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

## Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
474822919863.dkr.ecr.cn-latest northwest-1.amazonaws.com.cn/ xgboost-neo:<tag>		inference		

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec', region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca', region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch', region='cn-northwest-1', version='1.8.0', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
423003514399.dkr.e0.4t0- northwest-1.amazonaws.com.cn/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
423003514399.dkr.e0.4t0- northwest-1.amazonaws.com.cn/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
423003514399.dkr.edr.0t0- northwest-1.amazonaws.com.cn/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
423003514399.dkr.edr.0t0- northwest-1.amazonaws.com.cn/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
423003514399.dkr.edr.t0- northwest-1.amazonaws.com.cn/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
423003514399.dkr.edr.t0- northwest-1.amazonaws.com.cn/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.edr.3t1- northwest-1.amazonaws.com.cn/ pytorch-inference- eia:<tag>		eia	CPU	py3
727897471807.dkr.edr.2t0- northwest-1.amazonaws.com.cn/		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
pytorch-inference:<tag>				
727897471807.dkr.edr. <del>31</del> -northwest-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
727897471807.dkr.edr. <del>40</del> -northwest-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py3
727897471807.dkr.edr. <del>50</del> -northwest-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py3
727897471807.dkr.edr. <del>60</del> -northwest-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
727897471807.dkr.edr. <del>71</del> -northwest-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
727897471807.dkr.edr. <del>80</del> -northwest-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
727897471807.dkr.edr. <del>81</del> -northwest-1.amazonaws.com.cn/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
727897471807.dkr.edr. <del>20</del> -northwest-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.edr. <del>31</del> -northwest-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.edr. <del>40</del> -northwest-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
727897471807.dkr.ecr.cn-0-northwest-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py3
727897471807.dkr.ecr.cn-0-northwest-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py3, py36
727897471807.dkr.ecr.Z1-northwest-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py3, py36
727897471807.dkr.ecr.X1-northwest-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py3, py36
727897471807.dkr.ecr.X1-northwest-1.amazonaws.com.cn/pytorch-training:<tag>		training	CPU, GPU	py3, py36

## Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1-northwest-1.amazonaws.com.cn/randomcutforest:<tag>		inference, training		

## Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn', region='cn-northwest-1', version='0.23-1', image_scope='inference')
```

Registry path	Version	Job types (image scope)		
451049120500.dkr.ecr.cn-0.20.0-northwest-1.amazonaws.com.cn/		inference, training		

Registry path	Version	Job types (image scope)		
sagemaker-scikit-learn:<tag>				
451049120500.dkr.ecr.cn-0.23-1 northwest-1.amazonaws.com.cn/ sagemaker-scikit-learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation', region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ semantic-segmentation:<tag>		inference, training		

### Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq', region='cn-northwest-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1 northwest-1.amazonaws.com.cn/ seq2seq:<tag>		inference, training		

### Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark', region='cn-northwest-1', version='3.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
844356804704.dkr.ecr.cn-2.4 northwest-1.amazonaws.com.cn/		processing		

Registry path	Version	Job types (image scope)		
sagemaker-spark-processing:<tag>				
844356804704.dkr.ecr.cn-3.0 northwest-1.amazonaws.com.cn/ sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving', region='cn-northwest-1', version='2.4')
```

Registry path	Version	Job types (image scope)		
451049120500.dkr.ecr.cn-2.2 northwest-1.amazonaws.com.cn/ sagemaker-sparkml-serving:<tag>		inference		
451049120500.dkr.ecr.cn-2.4 northwest-1.amazonaws.com.cn/ sagemaker-sparkml-serving:<tag>		inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='cn-northwest-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
423003514399.dkr.ecr.t0.0 northwest-1.amazonaws.com.cn/ sagemaker-tensorflow-eia:<tag>		eia	CPU	py2
423003514399.dkr.ecr.t1.0 northwest-1.amazonaws.com.cn/ sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
423003514399.dkr.e <u>cr</u> .t <u>0</u> :0 northwest-1.amazonaws.com.cn/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
423003514399.dkr.e <u>cr</u> .t <u>0</u> :1 northwest-1.amazonaws.com.cn/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
423003514399.dkr.e <u>cr</u> .t <u>0</u> :0 northwest-1.amazonaws.com.cn/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
423003514399.dkr.e <u>cr</u> .t <u>0</u> :0 northwest-1.amazonaws.com.cn/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
423003514399.dkr.e <u>cr</u> .t <u>0</u> :0 northwest-1.amazonaws.com.cn/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
423003514399.dkr.e <u>cr</u> .t <u>0</u> :0 northwest-1.amazonaws.com.cn/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
423003514399.dkr.e <u>cr</u> .t <u>0</u> :0 northwest-1.amazonaws.com.cn/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
423003514399.dkr.e <u>cr</u> .t <u>0</u> :0 northwest-1.amazonaws.com.cn/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
423003514399.dkr.e <u>cr</u> .t <u>0</u> :0 northwest-1.amazonaws.com.cn/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
423003514399.dkr.edr. <b>41</b> -northwest-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
423003514399.dkr.edr. <b>41</b> -northwest-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
423003514399.dkr.edr. <b>50</b> -northwest-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
423003514399.dkr.edr. <b>50</b> -northwest-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
423003514399.dkr.edr. <b>60</b> -northwest-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
423003514399.dkr.edr. <b>60</b> -northwest-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
423003514399.dkr.edr. <b>70</b> -northwest-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
423003514399.dkr.edr. <b>70</b> -northwest-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
423003514399.dkr.edr. <b>80</b> -northwest-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
423003514399.dkr.edr. <b>80</b> -northwest-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
423003514399.dkr.edr. <b>90</b> -northwest-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
423003514399.dkr.e <u>8</u> . <u>10</u> -northwest-1.amazonaws.com.cn/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
727897471807.dkr.e <u>8</u> . <u>14</u> :0-northwest-1.amazonaws.com.cn/tensorflow-inference-eia:<tag>		eia	CPU	-
727897471807.dkr.e <u>8</u> . <u>15</u> :0-northwest-1.amazonaws.com.cn/tensorflow-inference-eia:<tag>		eia	CPU	-
727897471807.dkr.e <u>8</u> . <u>01</u> :0-northwest-1.amazonaws.com.cn/tensorflow-inference-eia:<tag>		eia	CPU	-
727897471807.dkr.e <u>8</u> . <u>01</u> :0-northwest-1.amazonaws.com.cn/tensorflow-inference-eia:<tag>		eia	CPU	-
727897471807.dkr.e <u>8</u> . <u>15</u> :0-northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>8</u> . <u>14</u> :0-northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>8</u> . <u>15</u> :0-northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>8</u> . <u>15</u> :2-northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>8</u> . <u>15</u> :3-northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
727897471807.dkr.e <u>0.0.4</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.0.5</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.0.6</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.0.7</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.0.8</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.0.9</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.0.10</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.0.11</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.0.12</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.0.13</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.0.14</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.1.0</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.1.1</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.1.2</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.1.3</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
727897471807.dkr.e <u>0.20</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.21</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.22</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.30</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.31</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.32</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.41</u> -northwest-1.amazonaws.com.cn/tensorflow-inference:<tag>		inference	CPU, GPU	-
727897471807.dkr.e <u>0.10</u> -northwest-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py3
727897471807.dkr.e <u>0.14</u> .0-northwest-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.e <u>0.15</u> .0-northwest-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.e <u>0.15</u> .2-northwest-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
727897471807.dkr.e <u>0.0.3</u> northwest-1.amazonaws.com.cn/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
727897471807.dkr.e <u>0.0.4</u> northwest-1.amazonaws.com.cn/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
727897471807.dkr.e <u>0.0.5</u> northwest-1.amazonaws.com.cn/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
727897471807.dkr.e <u>0.0.6</u> northwest-1.amazonaws.com.cn/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.e <u>0.0.7</u> northwest-1.amazonaws.com.cn/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.e <u>0.0.8</u> northwest-1.amazonaws.com.cn/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.e <u>0.0.9</u> northwest-1.amazonaws.com.cn/ tensorflow- training:<tag>		training	CPU, GPU	py3
727897471807.dkr.e <u>0.0.10</u> northwest-1.amazonaws.com.cn/ tensorflow- training:<tag>		training	CPU, GPU	py3
727897471807.dkr.e <u>0.0.11</u> northwest-1.amazonaws.com.cn/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
727897471807.dkr.e <u>0.0.12</u> northwest-1.amazonaws.com.cn/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
727897471807.dkr.e0.10-northwest-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py3
727897471807.dkr.e0.20-northwest-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py37
727897471807.dkr.e0.21-northwest-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py37
727897471807.dkr.e0.22-northwest-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py37
727897471807.dkr.e0.30-northwest-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py37
727897471807.dkr.e0.31-northwest-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py37
727897471807.dkr.e0.32-northwest-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py37
727897471807.dkr.e0.41-northwest-1.amazonaws.com.cn/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow',region='cn-northwest-1',version='1.0.0',image_scope='training',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
423003514399.dkr.e0.10-northwest-1.amazonaws.com.cn/sagemaker-rl-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow:coach0.10-<tag>				
423003514399.dkr.e0.10.1 northwest-1.amazonaws.com.cn/ sagemaker-rl- tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
423003514399.dkr.e0.11- northwest-1.amazonaws.com.cn/ sagemaker-rl- tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
423003514399.dkr.e0.11.0 northwest-1.amazonaws.com.cn/ sagemaker-rl- tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
423003514399.dkr.e0.11.1 northwest-1.amazonaws.com.cn/ sagemaker-rl- tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inference (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow', region='cn-northwest-1', version='1.15.0', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
474822919863.dkr.e0.15.0 northwest-1.amazonaws.com.cn/ sagemaker-neo- tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow', region='cn-northwest-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
423003514399.dkr.ecr.cn-northwest-1.amazonaws.com.cn/sagemaker-rl-tensorflow:ray0.5-<tag>		training	CPU, GPU	py3
423003514399.dkr.ecr.cn-northwest-1.amazonaws.com.cn/sagemaker-rl-tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
423003514399.dkr.ecr.cn-northwest-1.amazonaws.com.cn/sagemaker-rl-tensorflow:ray0.6-<tag>		training	CPU, GPU	py3
423003514399.dkr.ecr.cn-northwest-1.amazonaws.com.cn/sagemaker-rl-tensorflow:ray0.6.5-<tag>		training	CPU, GPU	py3

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost',region='cn-northwest-1',version='1.2-1')
```

Registry path	Version	Job types (image scope)		
387376663083.dkr.ecr.cn-1northwest-1.amazonaws.com.cn/xgboost:<tag>		inference, training		
451049120500.dkr.ecr.cn-0.90-1northwest-1.amazonaws.com.cn/sagemaker-xgboost:<tag>		inference, training		
451049120500.dkr.ecr.cn-0.90-2northwest-1.amazonaws.com.cn/sagemaker-xgboost:<tag>		inference, training		
451049120500.dkr.ecr.cn-1.0-1northwest-1.amazonaws.com.cn/sagemaker-xgboost:<tag>		inference, training		

Registry path	Version	Job types (image scope)		
451049120500.dkr.ecr.cn-1.2-1 northwest-1.amazonaws.com.cn/ sagemaker- xgboost:<tag>		inference, training		
451049120500.dkr.ecr.cn-1.2-2 northwest-1.amazonaws.com.cn/ sagemaker- xgboost:<tag>		inference, training		
451049120500.dkr.ecr.cn-1.3-1 northwest-1.amazonaws.com.cn/ sagemaker- xgboost:<tag>		inference, training		

### Docker Registry Paths and Example Code for Europe (Frankfurt) (eu-central-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

#### Topics

- [BlazingText \(algorithm\) \(p. 1111\)](#)
- [Chainer \(DLC\) \(p. 1111\)](#)
- [Clarify \(algorithm\) \(p. 1112\)](#)
- [Data Wrangler \(algorithm\) \(p. 1112\)](#)
- [Debugger \(algorithm\) \(p. 1112\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 1113\)](#)
- [Factorization Machines \(algorithm\) \(p. 1113\)](#)
- [Hugging Face \(algorithm\) \(p. 1113\)](#)
- [IP Insights \(algorithm\) \(p. 1114\)](#)
- [Image classification \(algorithm\) \(p. 1114\)](#)
- [Inferentia MXNet \(DLC\) \(p. 1115\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 1115\)](#)
- [K-Means \(algorithm\) \(p. 1115\)](#)
- [KNN \(algorithm\) \(p. 1115\)](#)
- [LDA \(algorithm\) \(p. 1116\)](#)
- [Linear Learner \(algorithm\) \(p. 1116\)](#)
- [MXNet \(DLC\) \(p. 1116\)](#)
- [MXNet Coach \(DLC\) \(p. 1119\)](#)
- [Model Monitor \(algorithm\) \(p. 1119\)](#)
- [NTM \(algorithm\) \(p. 1120\)](#)
- [Neo Image Classification \(algorithm\) \(p. 1120\)](#)
- [Neo MXNet \(DLC\) \(p. 1120\)](#)
- [Neo PyTorch \(DLC\) \(p. 1121\)](#)
- [Neo Tensorflow \(DLC\) \(p. 1121\)](#)
- [Neo XGBoost \(algorithm\) \(p. 1122\)](#)

- [Object Detection \(algorithm\) \(p. 1122\)](#)
- [Object2Vec \(algorithm\) \(p. 1122\)](#)
- [PCA \(algorithm\) \(p. 1122\)](#)
- [PyTorch \(DLC\) \(p. 1123\)](#)
- [Random Cut Forest \(algorithm\) \(p. 1125\)](#)
- [Ray PyTorch \(DLC\) \(p. 1125\)](#)
- [Scikit-learn \(algorithm\) \(p. 1126\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 1126\)](#)
- [Seq2Seq \(algorithm\) \(p. 1126\)](#)
- [Spark \(algorithm\) \(p. 1127\)](#)
- [SparkML Serving \(algorithm\) \(p. 1127\)](#)
- [Tensorflow \(DLC\) \(p. 1127\)](#)
- [Tensorflow Coach \(DLC\) \(p. 1135\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 1135\)](#)
- [Tensorflow Ray \(DLC\) \(p. 1136\)](#)
- [VW \(algorithm\) \(p. 1137\)](#)
- [XGBoost \(algorithm\) \(p. 1137\)](#)

### [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
813361260812.dkr.ecr.eu-central-1.amazonaws.com/blazingtext:<tag>		inference, training		

### [Chainer \(DLC\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer', region='eu-central-1', version='5.0.0', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-chainer:<tag>	4.0.0	inference, training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-chainer:<tag>	4.1.0	inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-chainer:<tag>	5.0.0	inference, training	CPU, GPU	py2, py3

### Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify', region='eu-central-1', version='1.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
017069133835.dkr.ecr.eu-1.0-central-1.amazonaws.com/sagemaker-clarify-processing:<tag>		processing		

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
024640144536.dkr.ecr.eu-1.x-central-1.amazonaws.com/sagemaker-data-wrangler-container:<tag>		processing		

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
482524230118.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-debugger-rules:<tag>		debugger		

### DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
495149712605.dkr.ecr.eu-central-1.amazonaws.com/forecasting-deepar:<tag>		inference, training		

### Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
664544806723.dkr.ecr.eu-central-1.amazonaws.com/factorization-machines:<tag>		inference, training		

### Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='eu-central-1', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.eu-central-1.amazonaws.com/	4.4.2	training		

Registry path	Version	Job types (image scope)		
huggingface-pytorch-training:<tag>				
763104351884.dkr.ecr.eu-4.5.0 central-1.amazonaws.com/ huggingface-pytorch-training:<tag>		training		
763104351884.dkr.ecr.eu-4.4.2 central-1.amazonaws.com/ huggingface-tensorflow-training:<tag>		training		
763104351884.dkr.ecr.eu-4.5.0 central-1.amazonaws.com/ huggingface-tensorflow-training:<tag>		training		

## IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
664544806723.dkr.ecr.eu-1 central-1.amazonaws.com/ ipinsights:<tag>		inference, training		

## Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
813361260812.dkr.ecr.eu-1 central-1.amazonaws.com/ image-classification:<tag>		inference, training		

## Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='eu-central-1',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
746233611703.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-neo-mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='eu-central-1',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
746233611703.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

## K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans',region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
664544806723.dkr.ecr.eu-central-1.amazonaws.com/kmeans:<tag>		inference, training		

## KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn',region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
664544806723.dkr.ecr.eu-central-1.amazonaws.com/knn:<tag>		inference, training		

### LDA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='lda', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
353608530281.dkr.ecr.eu-central-1.amazonaws.com/lda:<tag>		inference, training		

### Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
664544806723.dkr.ecr.eu-central-1.amazonaws.com/linear-learner:<tag>		inference, training		

### MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='eu-central-1', version='1.4.1', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>40</u> -central-1.amazonaws.com/sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.e <u>40</u> -central-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>41</u> -central-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e <u>0.12.1</u> -central-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.12.1</u> -central-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.00</u> -central-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.00</u> -central-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.10</u> -central-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.10</u> -central-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.20</u> -central-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.20</u> -central-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>5.0</u> -central-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>5.0</u> -central-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>5.0</u> -central-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>5.1</u> -central-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.e <u>5.1</u> -central-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.e <u>5.1</u> -central-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.e <u>5.0</u> -central-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.e <u>5.1</u> -central-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.e <u>5.0</u> -central-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.e <u>5.0</u> -central-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.e <u>5.0</u> -central-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>u</u> -central-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>u</u> -central-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>u</u> -central-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>u</u> -central-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

## MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='eu-central-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>u</u> -central-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e <u>u</u> -central-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

## Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
048819808253.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
664544806723.dkr.ecr.eu-central-1.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
746233611703.dkr.ecr.eu-central-1.amazonaws.com/image-classification-neo:<tag>	latest	inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet', region='eu-central-1', version='1.8', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
746233611703.dkr.ecr.eu-central-1.amazonaws.com/		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-inference-mxnet:<tag>				

## Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='eu-central-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
746233611703.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
746233611703.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
746233611703.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

## Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='eu-central-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
746233611703.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

## Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
746233611703.dkr.ecr.eu-central-1.amazonaws.com/xgboost-neo:<tag>		inference		

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
813361260812.dkr.ecr.eu-central-1.amazonaws.com/object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
664544806723.dkr.ecr.eu-central-1.amazonaws.com/object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
664544806723.dkr.ecr.eu-central-1.amazonaws.com/pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch', region='eu-central-1', version='1.8.0', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.40-central-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.40-central-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0-central-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0-central-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0-central-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0-central-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.3.1-central-1.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr.2.0-central-1.amazonaws.com/		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
pytorch-inference:<tag>				
763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.ecr.us-west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.ecr.us-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.ecr.eu-central-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.ecr.eu-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.ecr.eu-west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.ecr.eu-north-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.ecr.ap-southeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.ecr.ap-southeast-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.ecr.ap-northeast-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.eu-central-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.ecr.eu-central-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr.eu-central-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr.eu-central-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr.eu-central-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

### Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
664544806723.dkr.ecr.eu-central-1.amazonaws.com/randomcutforest:<tag>		inference, training		

### Ray PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-pytorch', region='eu-central-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.ecr.eu-central-1.amazonaws.com/		training	CPU, GPU	py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-rl-ray-container:ray-0.8.5-torch-<tag>				

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn',region='eu-central-1',version='0.23-1',image_scope='inference')
```

Registry path	Version	Job types (image scope)		
492215442770.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-scikit-learn:<tag>		inference, training		
492215442770.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-scikit-learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
813361260812.dkr.ecr.eu-central-1.amazonaws.com/semantic-segmentation:<tag>		inference, training		

### Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='eu-central-1')
```

Registry path	Version	Job types (image scope)		
813361260812.dkr.ecr.eu-1 central-1.amazonaws.com/ seq2seq:<tag>		inference, training		

## Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark', region='eu-central-1', version='3.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
906073651304.dkr.ecr.eu-2.4 central-1.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		
906073651304.dkr.ecr.eu-3.0 central-1.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving', region='eu-central-1', version='2.4')
```

Registry path	Version	Job types (image scope)		
492215442770.dkr.ecr.eu-2.2 central-1.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		
492215442770.dkr.ecr.eu-2.4 central-1.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='tensorflow', region='eu-central-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-tensorflow-eia:<tag>	10.0	eia	CPU	py2
520713654638.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>	11.0	training	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>	12.0	training	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>	13.1	training	CPU, GPU	py2
520713654638.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>	14.0	eia	CPU	-
520713654638.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>	14.1	eia	CPU	-
520713654638.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>	15.0	eia	CPU	-
520713654638.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-tensorflow-serving:<tag>	14.0	inference	CPU, GPU	-
520713654638.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-	14.0	inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow-serving:<tag>				
520713654638.dkr.ecr. <b>10.0</b> -central-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <b>10.0</b> -central-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr. <b>11</b> -central-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <b>11</b> -central-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr. <b>5.0</b> -central-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <b>5.0</b> -central-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr. <b>6.0</b> -central-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <b>6.0</b> -central-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr. <b>7.0</b> -central-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.ecr. <b>7.0</b> -central-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e-dr.8.0-central-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.8.0-central-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.e-dr.9.0-central-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.9.0-central-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.e-dr.14.0-central-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e-dr.15.0-central-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e-dr.16.0-central-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e-dr.17.0-central-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e-dr.18.0-central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e-dr.19.0-central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>15</u> .0 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>15</u> .2 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>15</u> .3 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>15</u> .4 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>15</u> .5 central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>20</u> .0- central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>20</u> .1- central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>20</u> .2- central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>20</u> .3- central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>20</u> .4- central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>20</u> .5- central-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.1</u> 1-central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.1</u> 2-central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.1</u> 3-central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.2</u> 0-central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.2</u> 1-central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.2</u> 2-central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.3</u> 0-central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.3</u> 1-central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.3</u> 2-central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.4</u> 1-central-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.4</u> 1-central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr. <b>14</b> .0 central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.ecr. <b>15</b> .0 central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.ecr. <b>15</b> .2 central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.ecr. <b>15</b> .3 central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.ecr. <b>15</b> .4 central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.ecr. <b>15</b> .5 central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.ecr. <b>0.0.0</b> - central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.ecr. <b>0.0.1</b> - central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.ecr. <b>0.0.2</b> - central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.ecr. <b>0.0.3</b> - central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.ecr. <b>0.0.4</b> - central-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.10</u> -central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.11</u> -central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.12</u> -central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.13</u> -central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.20</u> -central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.21</u> -central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.22</u> -central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.30</u> -central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.31</u> -central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.32</u> -central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.41</u> -central-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

## Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow',region='eu-central-1',version='1.0.0',image_scope='training',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.0-central-1.amazonaws.com/sagemaker-rl-coach-container:coach-1.0.0-tf-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10-central-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10.1-central-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11-central-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.0-central-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.1-central-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inferentia (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow',region='eu-central-1',version='1.15.0',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
746233611703.dkr.e <u>0.15.0</u> central-1.amazonaws.com/ sagemaker-neo- tensorflow:<tag>		inference	inf	py3

### Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow', region='eu-
central-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e <u>0.8.2</u> - central-1.amazonaws.com/ sagemaker-rl-ray- container:ray-0.8.2- tf-<tag>		training	CPU, GPU	py36
462105765813.dkr.e <u>0.8.5</u> - central-1.amazonaws.com/ sagemaker-rl-ray- container:ray-0.8.5- tf-<tag>		training	CPU, GPU	py36
520713654638.dkr.e <u>0.5.0</u> - central-1.amazonaws.com/ sagemaker-rl- tensorflow:ray0.5- <tag>		training	CPU, GPU	py3
520713654638.dkr.e <u>0.5.3</u> - central-1.amazonaws.com/ sagemaker-rl- tensorflow:ray0.5.3- <tag>		training	CPU, GPU	py3
520713654638.dkr.e <u>0.6.0</u> - central-1.amazonaws.com/ sagemaker-rl- tensorflow:ray0.6- <tag>		training	CPU, GPU	py3
520713654638.dkr.e <u>0.6.5</u> - central-1.amazonaws.com/ sagemaker-rl- tensorflow:ray0.6.5- <tag>		training	CPU, GPU	py3

## VW (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='vw', region='eu-central-1', version='8.7.0', image_scope='training')
```

Registry path	Version	Job types (image scope)		
462105765813.dkr.ecr.eu-8.7.0 central-1.amazonaws.com/ sagemaker-rl-vw- container:vw-8.7.0- <span style="color: red;">&lt;tag&gt;</span>		training		

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost', region='eu-central-1', version='1.2-1')
```

Registry path	Version	Job types (image scope)		
492215442770.dkr.ecr.eu-0.90-1 central-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
492215442770.dkr.ecr.eu-0.90-2 central-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
492215442770.dkr.ecr.eu-1.0-1 central-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
492215442770.dkr.ecr.eu-1.2-1 central-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
492215442770.dkr.ecr.eu-1.2-2 central-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
492215442770.dkr.ecr.eu-1.3-1 central-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		

Registry path	Version	Job types (image scope)		
813361260812.dkr.ecr.eu-1 central-1.amazonaws.com/ xgboost:<tag>		inference, training		

## Docker Registry Paths and Example Code for Europe (Ireland) (eu-west-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

### Topics

- [BlazingText \(algorithm\) \(p. 1139\)](#)
- [Chainer \(DLC\) \(p. 1139\)](#)
- [Clarify \(algorithm\) \(p. 1139\)](#)
- [Data Wrangler \(algorithm\) \(p. 1140\)](#)
- [Debugger \(algorithm\) \(p. 1140\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 1140\)](#)
- [Factorization Machines \(algorithm\) \(p. 1141\)](#)
- [Hugging Face \(algorithm\) \(p. 1141\)](#)
- [IP Insights \(algorithm\) \(p. 1142\)](#)
- [Image classification \(algorithm\) \(p. 1142\)](#)
- [Inferentia MXNet \(DLC\) \(p. 1142\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 1143\)](#)
- [K-Means \(algorithm\) \(p. 1143\)](#)
- [KNN \(algorithm\) \(p. 1143\)](#)
- [LDA \(algorithm\) \(p. 1143\)](#)
- [Linear Learner \(algorithm\) \(p. 1144\)](#)
- [MXNet \(DLC\) \(p. 1144\)](#)
- [MXNet Coach \(DLC\) \(p. 1147\)](#)
- [Model Monitor \(algorithm\) \(p. 1147\)](#)
- [NTM \(algorithm\) \(p. 1147\)](#)
- [Neo Image Classification \(algorithm\) \(p. 1148\)](#)
- [Neo MXNet \(DLC\) \(p. 1148\)](#)
- [Neo PyTorch \(DLC\) \(p. 1148\)](#)
- [Neo Tensorflow \(DLC\) \(p. 1149\)](#)
- [Neo XGBoost \(algorithm\) \(p. 1149\)](#)
- [Object Detection \(algorithm\) \(p. 1150\)](#)
- [Object2Vec \(algorithm\) \(p. 1150\)](#)
- [PCA \(algorithm\) \(p. 1150\)](#)
- [PyTorch \(DLC\) \(p. 1150\)](#)
- [Random Cut Forest \(algorithm\) \(p. 1153\)](#)
- [Ray PyTorch \(DLC\) \(p. 1153\)](#)
- [Scikit-learn \(algorithm\) \(p. 1153\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 1154\)](#)
- [Seq2Seq \(algorithm\) \(p. 1154\)](#)

- [Spark \(algorithm\) \(p. 1154\)](#)
- [SparkML Serving \(algorithm\) \(p. 1155\)](#)
- [Tensorflow \(DLC\) \(p. 1155\)](#)
- [Tensorflow Coach \(DLC\) \(p. 1162\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 1163\)](#)
- [Tensorflow Ray \(DLC\) \(p. 1163\)](#)
- [VW \(algorithm\) \(p. 1164\)](#)
- [XGBoost \(algorithm\) \(p. 1165\)](#)

### [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
685385470294.dkr.ecr.eu-1 west-1.amazonaws.com/ blazingtext:<tag>		inference, training		

### [Chainer \(DLC\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer',region='eu-
west-1',version='5.0.0',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-1 west-1.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-1 west-1.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-1 west-1.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3

### [Clarify \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify',region='eu-west-1',version='1.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
131013547314.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-clarify-processing:<tag>		processing		

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
245179582081.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-data-wrangler-container:<tag>		processing		

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
929884845733.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-debugger-rules:<tag>	latest	debugger		

### DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
224300973850.dkr.ecr.eu-1 west-1.amazonaws.com/ forecasting- deepar:<tag>		inference, training		

### Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
438346466558.dkr.ecr.eu-1 west-1.amazonaws.com/ factorization- machines:<tag>		inference, training		

### Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface',region='eu-west-1',version='4.4.2',image_scope='training',base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.eu-4.4.2 west-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.eu-4.5.0 west-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.eu-4.4.2 west-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		
763104351884.dkr.ecr.eu-4.5.0 west-1.amazonaws.com/ huggingface-		training		

Registry path	Version	Job types (image scope)		
tensorflow-training:<tag>				

### IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
438346466558.dkr.ecr.eu-west-1.amazonaws.com/ipinsights:<tag>		inference, training		

### Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
685385470294.dkr.ecr.eu-west-1.amazonaws.com/image-classification:<tag>		inference, training		

### Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='eu-west-1',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='eu-west-1',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

## K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
438346466558.dkr.ecr.eu-west-1.amazonaws.com/kmeans:<tag>		inference, training		

## KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
438346466558.dkr.ecr.eu-west-1.amazonaws.com/knn:<tag>		inference, training		

## LDA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='lda',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
999678624901.dkr.ecr.eu-west-1.amazonaws.com/lda:<tag>		inference, training		

### Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
438346466558.dkr.ecr.eu-west-1.amazonaws.com/linear-learner:<tag>		inference, training		

### MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='eu-west-1', version='1.4.1', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-mxnet-eia:<tag>	edr.3.0-	eia	CPU	py2, py3
520713654638.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-mxnet-serving-eia:<tag>	edr.4.0-	eia	CPU	py2, py3
520713654638.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-mxnet-serving:<tag>	edr.4.0-	inference	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-mxnet-serving:<tag>	edr.4.1-	inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0. <del>12</del> .1-west-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0. <del>12</del> .1-west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.00-west-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.00-west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.10-west-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.10-west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.20-west-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.20-west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.30-west-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.30-west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.40-west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.us-west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.ecr.us-west-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.ecr.us-west-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.ecr.us-west-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.ecr.us-west-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.ecr.us-west-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.ecr.us-west-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.ecr.us-west-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37
763104351884.dkr.ecr.us-west-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.ecr.us-west-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.ecr.us-west-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.eu-west-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

### MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='eu-west-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor', region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
468650794304.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='ntm', region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
438346466558.dkr.ecr.eu-west-1.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo', region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
802834080501.dkr.ecr.eu-west-1.amazonaws.com/image-classification-neo:<tag>	latest	inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet', region='eu-west-1', version='1.8', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='eu-west-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='eu-west-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
802834080501.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-inference-tensorflow:<tag>	1.15.3	inference	CPU, GPU	py3

### Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
802834080501.dkr.ecr.eu-west-1.amazonaws.com/xgboost-neo:<tag>	latest	inference		

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
685385470294.dkr.ecr.eu-1 west-1.amazonaws.com/ object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
438346466558.dkr.ecr.eu-1 west-1.amazonaws.com/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
438346466558.dkr.ecr.eu-1 west-1.amazonaws.com/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch',region='eu-
west-1',version='1.8.0',py_version='py3',image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.40-west-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.40-west-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0-west-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0-west-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0-west-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0-west-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.3.1-west-1.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr.2.0-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr.3.1-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr.4.0-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr.5.0-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <b>6.0</b> -west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <b>7.1</b> -west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <b>8.0</b> -west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <b>8.1</b> -west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <b>2.0</b> -west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>3.1</b> -west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>4.0</b> -west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>5.0</b> -west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <b>6.0</b> -west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <b>7.1</b> -west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <b>8.0</b> -west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.eu-west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

### Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
438346466558.dkr.ecr.eu-west-1.amazonaws.com/randomcutforest:<tag>		inference, training		

### Ray PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-pytorch', region='eu-west-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-torch-<tag>		training	CPU, GPU	py36

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn', region='eu-west-1', version='0.23-1', image_scope='inference')
```

Registry path	Version	Job types (image scope)		
141502667606.dkr.ecr.eu-west-1.amazonaws.com/	0.20.0	inference, training		

Registry path	Version	Job types (image scope)		
sagemaker-scikit-learn:<tag>				
141502667606.dkr.ecr.eu-0.23-1 west-1.amazonaws.com/ sagemaker-scikit-learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
685385470294.dkr.ecr.eu-1 west-1.amazonaws.com/ semantic-segmentation:<tag>		inference, training		

### Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='eu-west-1')
```

Registry path	Version	Job types (image scope)		
685385470294.dkr.ecr.eu-1 west-1.amazonaws.com/ seq2seq:<tag>		inference, training		

### Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='eu-west-1',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
571004829621.dkr.ecr.eu-2.4 west-1.amazonaws.com/		processing		

Registry path	Version	Job types (image scope)		
sagemaker-spark-processing:<tag>				
571004829621.dkr.ecr.eu-3.0 west-1.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving', region='eu-west-1', version='2.4')
```

Registry path	Version	Job types (image scope)		
141502667606.dkr.ecr.eu-2.2 west-1.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		
141502667606.dkr.ecr.eu-2.4 west-1.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='eu-west-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-0.0 west-1.amazonaws.com/ sagemaker-tensorflow-eia:<tag>		eia	CPU	py2
520713654638.dkr.ecr.eu-1.0 west-1.amazonaws.com/ sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e-dr.12.0 west-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e-dr.13.1 west-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
520713654638.dkr.e-dr.14.0 west-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.14.0 west-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.15.0 west-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.16.0 west-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.17.0 west-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.18.0 west-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.19.0 west-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <b>41</b> -west-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>41</b> -west-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>50</b> -west-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>50</b> -west-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>60</b> -west-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>60</b> -west-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>70</b> -west-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>70</b> -west-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>80</b> -west-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>80</b> -west-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>90</b> -west-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>9</u> .0-west-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.e <u>14</u> .0-west-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>15</u> .0-west-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>20</u> .0-west-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>20</u> .1-west-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>20</u> .2-west-1.amazonaws.com/tensorflow-inference-eia:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>24</u> .0-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>25</u> .0-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>25</u> .2-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>25</u> .3-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.15</u> .4 west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.15</u> .5 west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.16</u> - west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.17</u> - west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.18</u> - west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.19</u> - west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.20</u> - west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.21</u> - west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.22</u> - west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.23</u> - west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.24</u> - west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.25</u> - west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.26</u> - west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.27</u> - west-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.20</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.21</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.22</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.30</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.31</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.32</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.41</u> -west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.14.1</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.14.0</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.15.0</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.15.2</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.15.3</u> west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.e <u>0.15.4</u> west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0.15.5</u> west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0.16.0</u> west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.16.1</u> west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.16.2</u> west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.16.3</u> west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.16.4</u> west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.16.5</u> west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.13</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.20</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.21</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.22</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.30</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.31</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.32</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.41</u> -west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow',region='eu-west-1',version='1.0.0',image_scope='training',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e <u>0.0</u> -west-1.amazonaws.com/sagemaker-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
rl-coach-container:coach-1.0.0-tf-<tag>				
520713654638.dkr.e0.10-west-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.10.1-west-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11-west-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.0-west-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.1-west-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inferentia (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow',region='eu-west-1',version='1.15.0',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
802834080501.dkr.e0.15.0-west-1.amazonaws.com/sagemaker-neo-tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow', region='eu-west-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8.2-west-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.2-tf-<tag>		training	CPU, GPU	py36
462105765813.dkr.e0.8.5-west-1.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-tf-<tag>		training	CPU, GPU	py36
520713654638.dkr.e0.5.eu-west-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.5.3-west-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6.eu-west-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6.5-west-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5-<tag>		training	CPU, GPU	py3

## VW (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='vw', region='eu-west-1', version='8.7.0', image_scope='training')
```

Registry path	Version	Job types (image scope)		
462105765813.dkr.ecr.eu-west-1.amazonaws.com/		training		

Registry path	Version	Job types (image scope)		
sagemaker-rl-vw-container:vw-8.7.0-<tag>				

### XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost',region='eu-west-1',version='1.2-1')
```

Registry path	Version	Job types (image scope)		
141502667606.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-xgboost:<tag>	0.90-1	inference, training		
141502667606.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-xgboost:<tag>	0.90-2	inference, training		
141502667606.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-xgboost:<tag>	1.0-1	inference, training		
141502667606.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-xgboost:<tag>	1.2-1	inference, training		
141502667606.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-xgboost:<tag>	1.2-2	inference, training		
141502667606.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-xgboost:<tag>	1.3-1	inference, training		
685385470294.dkr.ecr.eu-west-1.amazonaws.com/xgboost:<tag>	1	inference, training		

### Docker Registry Paths and Example Code for Europe (London) (eu-west-2)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

## Topics

- [BlazingText \(algorithm\) \(p. 1167\)](#)
- [Chainer \(DLC\) \(p. 1167\)](#)
- [Clarify \(algorithm\) \(p. 1167\)](#)
- [Data Wrangler \(algorithm\) \(p. 1168\)](#)
- [Debugger \(algorithm\) \(p. 1168\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 1168\)](#)
- [Factorization Machines \(algorithm\) \(p. 1169\)](#)
- [Hugging Face \(algorithm\) \(p. 1169\)](#)
- [IP Insights \(algorithm\) \(p. 1170\)](#)
- [Image classification \(algorithm\) \(p. 1170\)](#)
- [Inferentia MXNet \(DLC\) \(p. 1170\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 1170\)](#)
- [K-Means \(algorithm\) \(p. 1171\)](#)
- [KNN \(algorithm\) \(p. 1171\)](#)
- [LDA \(algorithm\) \(p. 1171\)](#)
- [Linear Learner \(algorithm\) \(p. 1172\)](#)
- [MXNet \(DLC\) \(p. 1172\)](#)
- [MXNet Coach \(DLC\) \(p. 1174\)](#)
- [Model Monitor \(algorithm\) \(p. 1175\)](#)
- [NTM \(algorithm\) \(p. 1175\)](#)
- [Neo Image Classification \(algorithm\) \(p. 1175\)](#)
- [Neo MXNet \(DLC\) \(p. 1176\)](#)
- [Neo PyTorch \(DLC\) \(p. 1176\)](#)
- [Neo Tensorflow \(DLC\) \(p. 1177\)](#)
- [Neo XGBoost \(algorithm\) \(p. 1177\)](#)
- [Object Detection \(algorithm\) \(p. 1177\)](#)
- [Object2Vec \(algorithm\) \(p. 1178\)](#)
- [PCA \(algorithm\) \(p. 1178\)](#)
- [PyTorch \(DLC\) \(p. 1178\)](#)
- [Random Cut Forest \(algorithm\) \(p. 1180\)](#)
- [Ray PyTorch \(DLC\) \(p. 1181\)](#)
- [Scikit-learn \(algorithm\) \(p. 1181\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 1181\)](#)
- [Seq2Seq \(algorithm\) \(p. 1182\)](#)
- [Spark \(algorithm\) \(p. 1182\)](#)
- [SparkML Serving \(algorithm\) \(p. 1182\)](#)
- [Tensorflow \(DLC\) \(p. 1183\)](#)
- [Tensorflow Coach \(DLC\) \(p. 1190\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 1191\)](#)
- [Tensorflow Ray \(DLC\) \(p. 1191\)](#)
- [VW \(algorithm\) \(p. 1192\)](#)
- [XGBoost \(algorithm\) \(p. 1192\)](#)

## BlazingText (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext', region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-1 west-2.amazonaws.com/ blazingtext:<tag>		inference, training		

## Chainer (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer', region='eu-west-2', version='5.0.0', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>4.0.0</u> - west-2.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.e <u>4.1.0</u> - west-2.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.e <u>5.0.0</u> - west-2.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3

## Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify', region='eu-west-2', version='1.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
440796970383.dkr.ecr.eu-1.0 west-2.amazonaws.com/		processing		

Registry path	Version	Job types (image scope)		
sagemaker-clarify-processing:<tag>				

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler', region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
894491911112.dkr.ecr.eu-1.x west-2.amazonaws.com/ sagemaker- data-wrangler- container:<tag>		processing		

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger', region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
250201462417.dkr.ecr.eu-latest west-2.amazonaws.com/ sagemaker-debugger- rules:<tag>		debugger		

### DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar', region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-1 west-2.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
forecasting-deepar:<tag>				

## Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines', region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-1 west-2.amazonaws.com/ factorization- machines:<tag>		inference, training		

## Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='eu-west-2', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.eu-4.4.2 west-2.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.eu-4.5.0 west-2.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.eu-4.4.2 west-2.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		
763104351884.dkr.ecr.eu-4.5.0 west-2.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		

## IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights',region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-1 west-2.amazonaws.com/ ipinsights:<tag>		inference, training		

## Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification',region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-1 west-2.amazonaws.com/ image- classification:<tag>		inference, training		

## Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='eu-  
west-2',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
205493899709.dkr.ecr.eu-1 west-2.amazonaws.com/ sagemaker-neo- mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='eu-  
west-2',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
205493899709.dkr.ecr.eu-1-west-2.amazonaws.com/ sagemaker-neo-pytorch:<tag>		inference	inf	py3

### K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans', region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-1-west-2.amazonaws.com/ kmeans:<tag>		inference, training		

### KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn', region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-1-west-2.amazonaws.com/ knn:<tag>		inference, training		

### LDA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='lda', region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-1-west-2.amazonaws.com/ lda:<tag>		inference, training		

## Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-west-2.amazonaws.com/linear-learner:<tag>		inference, training		

## MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='eu-west-2', version='1.4.1', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr.10-west-2.amazonaws.com/sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.edr.10-west-2.amazonaws.com/sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.edr.10-west-2.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.11-west-2.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e0.10.1-west-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.10.1-west-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <b>00</b> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>00</b> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>10</b> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>10</b> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>20</b> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>20</b> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>30</b> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>30</b> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>40</b> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>40</b> -west-2.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <b>41</b> -west-2.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <del>51</del> -west-2.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <del>70</del> -west-2.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr. <del>41</del> -west-2.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>60</del> -west-2.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>70</del> -west-2.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <del>80</del> -west-2.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37
763104351884.dkr.edr. <del>41</del> -west-2.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>60</del> -west-2.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>70</del> -west-2.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>80</del> -west-2.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

## MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='coach-mxnet',region='eu-west-2',version='0.11',py_version='py3',image_scope='training',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.eu-west-2.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.eu-west-2.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor',region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
749857270468.dkr.ecr.eu-west-2.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm',region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-west-2.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='image-classification-neo',region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
205493899709.dkr.ecr.eu-west-2.amazonaws.com/image-classification-neo:<tag>	latest	inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet',region='eu-west-2',version='1.8',py_version='py3',image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
205493899709.dkr.ecr.eu-west-2.amazonaws.com/sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch',region='eu-west-2',version='1.6',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
205493899709.dkr.ecr.eu-west-2.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
205493899709.dkr.ecr.eu-west-2.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
205493899709.dkr.ecr.eu-west-2.amazonaws.com/		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-inference-pytorch:<tag>				

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='eu-west-2', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
205493899709.dkr.ecr.eu-west-2.amazonaws.com/sagemaker-inference-tensorflow:<tag>	1.15.3	inference	CPU, GPU	py3

### Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
205493899709.dkr.ecr.eu-west-2.amazonaws.com/xgboost-neo:<tag>	latest	inference		

### Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-west-2.amazonaws.com/object-detection:<tag>	1	inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec',region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-1 west-2.amazonaws.com/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca',region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-1 west-2.amazonaws.com/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch',region='eu-
west-2',version='1.8.0',py_version='py3',image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.40- west-2.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.40- west-2.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e0.40- west-2.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <b>00</b> -west-2.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>10</b> -west-2.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>10</b> -west-2.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>01</b> -west-2.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr. <b>20</b> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>30</b> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>40</b> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <b>50</b> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <b>60</b> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <b>70</b> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <b>80</b> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <del>81</del> -west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>20</del> -west-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>51</del> -west-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>40</del> -west-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>50</del> -west-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>60</del> -west-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>71</del> -west-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>80</del> -west-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <del>81</del> -west-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

### Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-1 west-2.amazonaws.com/ randomcutforest:<tag>		inference, training		

### Ray PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-pytorch', region='eu-west-2', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8.5- west-2.amazonaws.com/ sagemaker-rl-ray- container:ray-0.8.5- torch-<tag>	0.8.5-	training	CPU, GPU	py36

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn', region='eu-west-2', version='0.23-1', image_scope='inference')
```

Registry path	Version	Job types (image scope)		
764974769150.dkr.ecr.eu-0.20.0 west-2.amazonaws.com/ sagemaker-scikit- learn:<tag>	-0.20.0	inference, training		
764974769150.dkr.ecr.eu-0.23-1 west-2.amazonaws.com/ sagemaker-scikit- learn:<tag>	-0.23-1	inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='semantic-segmentation',region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-1 west-2.amazonaws.com/ semantic- segmentation:<tag>		inference, training		

## Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='eu-west-2')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-1 west-2.amazonaws.com/ seq2seq:<tag>		inference, training		

## Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='eu-  
west-2',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
836651553127.dkr.ecr.eu-2.4 west-2.amazonaws.com/ sagemaker-spark- processing:<tag>		processing		
836651553127.dkr.ecr.eu-3.0 west-2.amazonaws.com/ sagemaker-spark- processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving',region='eu-west-2',version='2.4')
```

Registry path	Version	Job types (image scope)		
764974769150.dkr.ecr.eu-2.2 west-2.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		
764974769150.dkr.ecr.eu-2.4 west-2.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		

### Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='eu-west-2', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-10.0 west-2.amazonaws.com/ sagemaker- tensorflow- eia:<tag>		eia	CPU	py2
520713654638.dkr.ecr.eu-11.0 west-2.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-12.0 west-2.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-13.1 west-2.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr.eu-14.0 west-2.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.ecr.eu-14.0 west-2.amazonaws.com/		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-tensorflow-serving-eia:<tag>				
520713654638.dkr.edr.111.0 west-2.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>		eia	CPU	-
520713654638.dkr.edr.111.0 west-2.amazonaws.com/sagemaker-tensorflow-serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.edr.121.0 west-2.amazonaws.com/sagemaker-tensorflow-serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.edr.10.0 west-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr.10.0 west-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr.41-west-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr.41-west-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr.50-west-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr.50-west-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr.60-west-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <del>6.0</del> -west-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> -west-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> -west-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> -west-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> -west-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>9.0</del> -west-2.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>9.0</del> -west-2.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <del>14.0</del> -west-2.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <del>15.0</del> -west-2.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <del>16.0</del> -west-2.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>2.30</u> -west-2.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>2.31</u> .0-west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.31</u> .0-west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.31</u> .0-west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.31</u> .2-west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.31</u> .3-west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.31</u> .4-west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.31</u> .5-west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.30</u> .0-west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.30</u> .1-west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.30</u> .2-west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.03</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.04</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.10</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.11</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.12</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.13</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.20</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.21</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.22</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.30</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.31</u> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <del>0.30.2</del> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.40.1</del> -west-2.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.edr. <del>1.0.1</del> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>1.0.0</del> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>1.5.0</del> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>1.5.2</del> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.edr. <del>1.5.3</del> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.edr. <del>1.5.4</del> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.edr. <del>1.5.5</del> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <del>0.0.0</del> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <del>0.0.1</del> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.02</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.03</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.04</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.10</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.11</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.12</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.13</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.20</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.21</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.22</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.30</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.10</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.10</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.11</u> -west-2.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow', region='eu-west-2', version='1.0.0', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e <u>0.10</u> -west-2.amazonaws.com/sagemaker-rl-coach-container:coach-1.0.0-tf-<tag>		training	CPU, GPU	py3
520713654638.dkr.e <u>0.10</u> -west-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
520713654638.dkr.e <u>0.10</u> .1-west-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e <u>0.11</u> -west-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e <u>0.11</u> .0-west-2.amazonaws.com/sagemaker-rl-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow:coach0.11.0-<tag>				
520713654638.dkr.e0.11.1-west-2.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inference (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow',region='eu-west-2',version='1.15.0',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
205493899709.dkr.e0.15.0-west-2.amazonaws.com/sagemaker-neo-tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow',region='eu-west-2',version='0.8.5',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
462105765813.dkr.e0.8.2-west-2.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.2-tf-<tag>		training	CPU, GPU	py36
462105765813.dkr.e0.8.5-west-2.amazonaws.com/sagemaker-rl-ray-container:ray-0.8.5-tf-<tag>		training	CPU, GPU	py36
520713654638.dkr.e0.5u-west-2.amazonaws.com/sagemaker-rl-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow:ray0.5-<tag>				
520713654638.dkr.e0.5.3-west-2.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6-eu-west-2.amazonaws.com/sagemaker-rl-tensorflow:ray0.6-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6.5-west-2.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5-<tag>		training	CPU, GPU	py3

## VW (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='vw', region='eu-west-2', version='8.7.0', image_scope='training')
```

Registry path	Version	Job types (image scope)		
462105765813.dkr.ecr.eu-west-2.amazonaws.com/sagemaker-rl-vw-container:vw-8.7.0-<tag>	8.7.0	training		

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost', region='eu-west-2', version='1.2-1')
```

Registry path	Version	Job types (image scope)		
644912444149.dkr.ecr.eu-west-2.amazonaws.com/xgboost:<tag>	1	inference, training		

Registry path	Version	Job types (image scope)		
764974769150.dkr.ecr.eu-west-2.amazonaws.com/sagemaker-xgboost:<tag>	0.90-1	inference, training		
764974769150.dkr.ecr.eu-west-2.amazonaws.com/sagemaker-xgboost:<tag>	0.90-2	inference, training		
764974769150.dkr.ecr.eu-west-2.amazonaws.com/sagemaker-xgboost:<tag>	1.0-1	inference, training		
764974769150.dkr.ecr.eu-west-2.amazonaws.com/sagemaker-xgboost:<tag>	1.2-1	inference, training		
764974769150.dkr.ecr.eu-west-2.amazonaws.com/sagemaker-xgboost:<tag>	1.2-2	inference, training		
764974769150.dkr.ecr.eu-west-2.amazonaws.com/sagemaker-xgboost:<tag>	1.3-1	inference, training		

### Docker Registry Paths and Example Code for Europe (Paris) (eu-west-3)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

#### Topics

- [BlazingText \(algorithm\) \(p. 1194\)](#)
- [Chainer \(DLC\) \(p. 1194\)](#)
- [Clarify \(algorithm\) \(p. 1195\)](#)
- [Data Wrangler \(algorithm\) \(p. 1195\)](#)
- [Debugger \(algorithm\) \(p. 1196\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 1196\)](#)
- [Factorization Machines \(algorithm\) \(p. 1196\)](#)
- [Hugging Face \(algorithm\) \(p. 1196\)](#)
- [IP Insights \(algorithm\) \(p. 1197\)](#)
- [Image classification \(algorithm\) \(p. 1197\)](#)
- [Inferentia MXNet \(DLC\) \(p. 1198\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 1198\)](#)
- [K-Means \(algorithm\) \(p. 1198\)](#)
- [KNN \(algorithm\) \(p. 1198\)](#)

- [Linear Learner \(algorithm\) \(p. 1199\)](#)
- [MXNet \(DLC\) \(p. 1199\)](#)
- [MXNet Coach \(DLC\) \(p. 1202\)](#)
- [Model Monitor \(algorithm\) \(p. 1202\)](#)
- [NTM \(algorithm\) \(p. 1202\)](#)
- [Neo Image Classification \(algorithm\) \(p. 1203\)](#)
- [Neo MXNet \(DLC\) \(p. 1203\)](#)
- [Neo PyTorch \(DLC\) \(p. 1203\)](#)
- [Neo Tensorflow \(DLC\) \(p. 1204\)](#)
- [Neo XGBoost \(algorithm\) \(p. 1204\)](#)
- [Object Detection \(algorithm\) \(p. 1205\)](#)
- [Object2Vec \(algorithm\) \(p. 1205\)](#)
- [PCA \(algorithm\) \(p. 1205\)](#)
- [PyTorch \(DLC\) \(p. 1205\)](#)
- [Random Cut Forest \(algorithm\) \(p. 1208\)](#)
- [Scikit-learn \(algorithm\) \(p. 1208\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 1208\)](#)
- [Seq2Seq \(algorithm\) \(p. 1209\)](#)
- [Spark \(algorithm\) \(p. 1209\)](#)
- [SparkML Serving \(algorithm\) \(p. 1209\)](#)
- [Tensorflow \(DLC\) \(p. 1210\)](#)
- [Tensorflow Coach \(DLC\) \(p. 1217\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 1218\)](#)
- [Tensorflow Ray \(DLC\) \(p. 1218\)](#)
- [XGBoost \(algorithm\) \(p. 1219\)](#)

### [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext', region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-west-3.amazonaws.com/blazingtext:<tag>		inference, training		

### [Chainer \(DLC\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='chainer', region='eu-west-3', version='5.0.0', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-chainer:<tag>	4.0.0	inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-chainer:<tag>	4.1.0	inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-chainer:<tag>	5.0.0	inference, training	CPU, GPU	py2, py3

### Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify', region='eu-west-3', version='1.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
341593696636.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-clarify-processing:<tag>	1.0	processing		

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler', region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
807237891255.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-data-wrangler-container:<tag>	1.x	processing		

## Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger', region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
447278800020.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-debugger-rules:<tag>	latest	debugger		

## DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar', region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-west-3.amazonaws.com/forecasting-deepar:<tag>	1	inference, training		

## Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines', region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-west-3.amazonaws.com/factorization-machines:<tag>	1	inference, training		

## Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='eu-west-3', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.eu-4.4.2 west-3.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.eu-4.5.0 west-3.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.eu-4.4.2 west-3.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		
763104351884.dkr.ecr.eu-4.5.0 west-3.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		

## IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights', region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-1 west-3.amazonaws.com/ ipinsights:<tag>		inference, training		

## Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification', region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-1 west-3.amazonaws.com/ image- classification:<tag>		inference, training		

## Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='eu-west-3',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
254080097072.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-neo-mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='eu-west-3',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
254080097072.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

## K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans',region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-west-3.amazonaws.com/kmeans:<tag>		inference, training		

## KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn',region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-1 west-3.amazonaws.com/ knn:<tag>		inference, training		

### Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-1 west-3.amazonaws.com/ linear-learner:<tag>		inference, training		

### MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='eu-west-3', version='1.4.1', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-1 west-3.amazonaws.com/ sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.ecr.eu-1 west-3.amazonaws.com/ sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.ecr.eu-1 west-3.amazonaws.com/ sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-1 west-3.amazonaws.com/ sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0. <del>12</del> .1-west-3.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0. <del>12</del> .1-west-3.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.00-west-3.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.00-west-3.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.10-west-3.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.10-west-3.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.20-west-3.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.20-west-3.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.30-west-3.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.30-west-3.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.40-west-3.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.us-west-3.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.ecr.us-west-3.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.ecr.us-west-3.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.ecr.us-west-3.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.ecr.us-west-3.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.ecr.us-west-3.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.ecr.us-west-3.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.ecr.us-west-3.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37
763104351884.dkr.ecr.us-west-3.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.ecr.us-west-3.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.ecr.us-west-3.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.eu-west-3.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

### MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='eu-west-3', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor', region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
680080141114.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='ntm', region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-west-3.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo', region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
254080097072.dkr.ecr.eu-west-3.amazonaws.com/image-classification-neo:<tag>	latest	inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet', region='eu-west-3', version='1.8', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
254080097072.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='eu-west-3', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
254080097072.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
254080097072.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
254080097072.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='eu-west-3', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
254080097072.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-inference-tensorflow:<tag>	1.15.3	inference	CPU, GPU	py3

### Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
254080097072.dkr.ecr.eu-west-3.amazonaws.com/xgboost-neo:<tag>	latest	inference		

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection',region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-1 west-3.amazonaws.com/ object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec',region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-1 west-3.amazonaws.com/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca',region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-1 west-3.amazonaws.com/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch',region='eu-
west-3',version='1.8.0',py_version='py3',image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.40-west-3.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.40-west-3.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0-west-3.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0-west-3.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0-west-3.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0-west-3.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.3.1-west-3.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr.2.0-west-3.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr.3.1-west-3.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr.4.0-west-3.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr.5.0-west-3.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <b>6.0</b> -west-3.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <b>7.1</b> -west-3.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <b>8.0</b> -west-3.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <b>8.1</b> -west-3.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.edr. <b>2.0</b> -west-3.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>3.1</b> -west-3.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>4.0</b> -west-3.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>5.0</b> -west-3.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <b>6.0</b> -west-3.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <b>7.1</b> -west-3.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.edr. <b>8.0</b> -west-3.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.eu-west-3.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

### Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest',region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-west-3.amazonaws.com/randomcutforest:<tag>		inference, training		

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn',region='eu-west-3',version='0.23-1',image_scope='inference')
```

Registry path	Version	Job types (image scope)		
659782779980.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-scikit-learn:<tag>	0.20.0	inference, training		
659782779980.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-scikit-learn:<tag>	0.23-1	inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-1 west-3.amazonaws.com/ semantic-segmentation:<tag>		inference, training		

## Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='eu-west-3')
```

Registry path	Version	Job types (image scope)		
749696950732.dkr.ecr.eu-1 west-3.amazonaws.com/ seq2seq:<tag>		inference, training		

## Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='eu-west-3',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
136845547031.dkr.ecr.eu-2.4 west-3.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		
136845547031.dkr.ecr.eu-3.0 west-3.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving',region='eu-west-3',version='2.4')
```

Registry path	Version	Job types (image scope)		
659782779980.dkr.ecr.eu-2.2 west-3.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		
659782779980.dkr.ecr.eu-2.4 west-3.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='eu-west-3', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-10.0 west-3.amazonaws.com/ sagemaker- tensorflow- eia:<tag>		eia	CPU	py2
520713654638.dkr.ecr.eu-11.0 west-3.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-12.0 west-3.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-13.1 west-3.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
520713654638.dkr.ecr.eu-14.0 west-3.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.ecr.eu-14.0 west-3.amazonaws.com/		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-tensorflow-serving-eia:<tag>				
520713654638.dkr.edr.111.0 west-3.amazonaws.com/ sagemaker-tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.edr.111.0 west-3.amazonaws.com/ sagemaker-tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.edr.121.0 west-3.amazonaws.com/ sagemaker-tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.edr.130.0 west-3.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr.130.0 west-3.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr.411- west-3.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr.411- west-3.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr.510- west-3.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr.510- west-3.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr.610- west-3.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <del>6.0</del> -west-3.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> -west-3.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>7.0</del> -west-3.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> -west-3.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>8.0</del> -west-3.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <del>9.0</del> -west-3.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <del>9.0</del> -west-3.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <del>14.0</del> -west-3.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <del>15.0</del> -west-3.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.edr. <del>16.0</del> -west-3.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>2.3.0</u> -west-3.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>2.4.0</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.4.0</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.5.0</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.5.2</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.5.3</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.5.4</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.5.5</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.6.0</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.6.1</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>2.6.2</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.03</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.04</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.10</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.11</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.12</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.13</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.20</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.21</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.22</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.30</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.31</u> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <del>0.30.2</del> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <del>0.40.1</del> -west-3.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.edr. <del>1.0.1</del> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>1.0.0</del> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>1.5.0</del> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>1.5.2</del> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.edr. <del>1.5.3</del> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.edr. <del>1.5.4</del> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.edr. <del>1.5.5</del> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <del>0.0.0</del> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <del>0.0.1</del> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.02</u> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.03</u> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.04</u> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.10</u> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.11</u> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.12</u> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.13</u> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.20</u> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.21</u> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.22</u> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.30</u> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.10</u> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.10.1</u> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.11</u> -west-3.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow', region='eu-west-3', version='1.0.0', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>0.10</u> -west-3.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
520713654638.dkr.e <u>0.10.1</u> -west-3.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e <u>0.11</u> -west-3.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.e <u>0.11.0</u> -west-3.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
520713654638.dkr.e <u>0.11.1</u> -west-3.amazonaws.com/sagemaker-rl-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow:coach0.11.1- <i>&lt;tag&gt;</i>				

## Tensorflow Inference (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow', region='eu-west-3', version='1.15.0', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
254080097072.dkr.e0r.15.0 west-3.amazonaws.com/ sagemaker-neo-tensorflow: <i>&lt;tag&gt;</i>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow', region='eu-west-3', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0r.5eu-west-3.amazonaws.com/ sagemaker-rl-tensorflow:ray0.5- <i>&lt;tag&gt;</i>		training	CPU, GPU	py3
520713654638.dkr.e0r.5.3- west-3.amazonaws.com/ sagemaker-rl-tensorflow:ray0.5.3- <i>&lt;tag&gt;</i>		training	CPU, GPU	py3
520713654638.dkr.e0r.6eu-west-3.amazonaws.com/ sagemaker-rl-tensorflow:ray0.6- <i>&lt;tag&gt;</i>		training	CPU, GPU	py3
520713654638.dkr.e0r.6.5- west-3.amazonaws.com/ sagemaker-rl-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow:ray0.6.5-<tag>				

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost', region='eu-west-3', version='1.2-1')
```

Registry path	Version	Job types (image scope)		
659782779980.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
659782779980.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
659782779980.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
659782779980.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
659782779980.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
659782779980.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
749696950732.dkr.ecr.eu-west-3.amazonaws.com/xgboost:<tag>		inference, training		

## Docker Registry Paths and Example Code for Europe (Stockholm) (eu-north-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

### Topics

- [BlazingText \(algorithm\) \(p. 1220\)](#)
- [Chainer \(DLC\) \(p. 1221\)](#)
- [Clarify \(algorithm\) \(p. 1221\)](#)
- [Data Wrangler \(algorithm\) \(p. 1222\)](#)
- [Debugger \(algorithm\) \(p. 1222\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 1222\)](#)
- [Factorization Machines \(algorithm\) \(p. 1222\)](#)
- [Hugging Face \(algorithm\) \(p. 1223\)](#)
- [IP Insights \(algorithm\) \(p. 1223\)](#)
- [Image classification \(algorithm\) \(p. 1224\)](#)
- [Inferentia MXNet \(DLC\) \(p. 1224\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 1224\)](#)
- [K-Means \(algorithm\) \(p. 1224\)](#)
- [KNN \(algorithm\) \(p. 1225\)](#)
- [Linear Learner \(algorithm\) \(p. 1225\)](#)
- [MXNet \(DLC\) \(p. 1225\)](#)
- [MXNet Coach \(DLC\) \(p. 1228\)](#)
- [Model Monitor \(algorithm\) \(p. 1228\)](#)
- [NTM \(algorithm\) \(p. 1229\)](#)
- [Neo Image Classification \(algorithm\) \(p. 1229\)](#)
- [Neo MXNet \(DLC\) \(p. 1229\)](#)
- [Neo PyTorch \(DLC\) \(p. 1230\)](#)
- [Neo Tensorflow \(DLC\) \(p. 1230\)](#)
- [Neo XGBoost \(algorithm\) \(p. 1231\)](#)
- [Object Detection \(algorithm\) \(p. 1231\)](#)
- [Object2Vec \(algorithm\) \(p. 1231\)](#)
- [PCA \(algorithm\) \(p. 1231\)](#)
- [PyTorch \(DLC\) \(p. 1232\)](#)
- [Random Cut Forest \(algorithm\) \(p. 1234\)](#)
- [Scikit-learn \(algorithm\) \(p. 1234\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 1235\)](#)
- [Seq2Seq \(algorithm\) \(p. 1235\)](#)
- [Spark \(algorithm\) \(p. 1235\)](#)
- [SparkML Serving \(algorithm\) \(p. 1236\)](#)
- [Tensorflow \(DLC\) \(p. 1236\)](#)
- [Tensorflow Coach \(DLC\) \(p. 1243\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 1244\)](#)
- [Tensorflow Ray \(DLC\) \(p. 1244\)](#)
- [XGBoost \(algorithm\) \(p. 1245\)](#)

## [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext', region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-north-1.amazonaws.com/blazingtext:<tag>		inference, training		

### Chainer (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer', region='eu-north-1', version='5.0.0', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-north-1.amazonaws.com/sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-north-1.amazonaws.com/sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.ecr.eu-north-1.amazonaws.com/sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3

### Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify', region='eu-north-1', version='1.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
763603941244.dkr.ecr.eu-1.0-north-1.amazonaws.com/sagemaker-clarify-processing:<tag>		processing		

## Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler',region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
054986407534.dkr.ecr.eu-1.x north-1.amazonaws.com/ sagemaker- data-wrangler- container:<tag>		processing		

## Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger',region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
314864569078.dkr.ecr.eu-latest north-1.amazonaws.com/ sagemaker-debugger- rules:<tag>		debugger		

## DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar',region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-1 north-1.amazonaws.com/ forecasting- deepar:<tag>		inference, training		

## Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines',region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-1 north-1.amazonaws.com/ factorization- machines:<tag>		inference, training		

## Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='eu-north-1', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.eu-4.4.2 north-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.eu-4.5.0 north-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.eu-4.4.2 north-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		
763104351884.dkr.ecr.eu-4.5.0 north-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		

## IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights', region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-1 north-1.amazonaws.com/ ipinsights:<tag>		inference, training		

## Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification', region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-1-north-1.amazonaws.com/image-classification:<tag>		inference, training		

## Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet', region='eu-north-1', version='1.5.1', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
601324751636.dkr.ecr.eu-1-north-1.amazonaws.com/sagemaker-neo-mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch', region='eu-north-1', version='1.5.1', py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
601324751636.dkr.ecr.eu-1-north-1.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

## K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='kmeans', region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-1 north-1.amazonaws.com/ kmeans:<tag>		inference, training		

### KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn', region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-1 north-1.amazonaws.com/ knn:<tag>		inference, training		

### Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-1 north-1.amazonaws.com/ linear-learner:<tag>		inference, training		

### MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='eu-north-1', version='1.4.1', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-1 north-1.amazonaws.com/		eia	CPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-mxnet-eia:<tag>				
520713654638.dkr.e <u>10</u> -north-1.amazonaws.com/sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.e <u>10</u> -north-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>11</u> -north-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e <u>0.12.1</u> -north-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.12.1</u> -north-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.10</u> -north-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.10</u> -north-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.10</u> -north-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.10</u> -north-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>0.21</u> -north-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <b>21</b> -north-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>30</b> -north-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>30</b> -north-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>40</b> -north-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr. <b>41</b> -north-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.edr. <b>41</b> -north-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <b>51</b> -north-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.edr. <b>70</b> -north-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.edr. <b>41</b> -north-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.edr. <b>60</b> -north-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.edr. <b>70</b> -north-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.edr. <del>2.0</del> -north-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37
763104351884.dkr.edr. <del>4.1</del> -north-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>6.0</del> -north-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <del>7.0</del> -north-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <del>8.0</del> -north-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

## MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='eu-north-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <del>0.11</del> -north-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
520713654638.dkr.edr. <del>0.11.0</del> -north-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

## Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='model-monitor',region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
895015795356.dkr.ecr.eu-north-1.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm',region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-north-1.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo',region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
601324751636.dkr.ecr.eu-north-1.amazonaws.com/image-classification-neo:<tag>	latest	inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet',region='eu-north-1',version='1.8',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
601324751636.dkr.e <u>u</u> -north-1.amazonaws.com/sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='eu-north-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
601324751636.dkr.e <u>u</u> -north-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
601324751636.dkr.e <u>u</u> -north-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
601324751636.dkr.e <u>u</u> -north-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='eu-north-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
601324751636.dkr.e <u>u</u> .3-north-1.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

## Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
601324751636.dkr.ecr.eu-north-1.amazonaws.com/xgboost-neo:<tag>		inference		

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-north-1.amazonaws.com/object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec', region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-north-1.amazonaws.com/object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca', region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-1 north-1.amazonaws.com/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch', region='eu-north-1', version='1.8.0', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.40- north-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.40- north-1.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0- north-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.0.0- north-1.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0- north-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.edr.1.0- north-1.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr.3.1- north-1.amazonaws.com/ pytorch-inference- eia:<tag>		eia	CPU	py3
763104351884.dkr.edr.2.0- north-1.amazonaws.com/		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
pytorch-inference:<tag>				
763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.ecr.us-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.ecr.us-west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.ecr.eu-central-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.ecr.eu-north-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.ecr.eu-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.ecr.eu-west-2.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.ecr.us-east-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.ecr.us-west-2.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr.eu-north-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.ecr.eu-north-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr.eu-north-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr.eu-north-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr.eu-north-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

## Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-north-1.amazonaws.com/randomcutforest:<tag>		inference, training		

## Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn', region='eu-north-1', version='0.23-1', image_scope='inference')
```

Registry path	Version	Job types (image scope)		
662702820516.dkr.ecr.eu-0.20.0-north-1.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
sagemaker-scikit-learn:<tag>				
662702820516.dkr.ecr.eu-0.23-1 north-1.amazonaws.com/ sagemaker-scikit-learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation', region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-1 north-1.amazonaws.com/ semantic-segmentation:<tag>		inference, training		

### Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq', region='eu-north-1')
```

Registry path	Version	Job types (image scope)		
669576153137.dkr.ecr.eu-1 north-1.amazonaws.com/ seq2seq:<tag>		inference, training		

### Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark', region='eu-north-1', version='3.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
330188676905.dkr.ecr.eu-2.4 north-1.amazonaws.com/		processing		

Registry path	Version	Job types (image scope)		
sagemaker-spark-processing:<tag>				
330188676905.dkr.ecr.eu-3.0 north-1.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving', region='eu-north-1', version='2.4')
```

Registry path	Version	Job types (image scope)		
662702820516.dkr.ecr.eu-2.2 north-1.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		
662702820516.dkr.ecr.eu-2.4 north-1.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='eu-north-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.eu-10.0 north-1.amazonaws.com/ sagemaker-tensorflow-eia:<tag>		eia	CPU	py2
520713654638.dkr.ecr.eu-11.0 north-1.amazonaws.com/ sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>12</u> .0 north-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e <u>13</u> .1 north-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
520713654638.dkr.e <u>14</u> .0 north-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e <u>14</u> .0 north-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e <u>15</u> .0 north-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e <u>16</u> .0 north-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e <u>17</u> .0 north-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e <u>18</u> .0 north-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e <u>19</u> .0 north-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.edr. <b>4.1</b> -north-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>4.1</b> -north-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>5.0</b> -north-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>5.0</b> -north-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>6.0</b> -north-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>6.0</b> -north-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>7.0</b> -north-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>7.0</b> -north-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>8.0</b> -north-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.edr. <b>8.0</b> -north-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.edr. <b>9.0</b> -north-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr. <b>9.0</b> -north-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.ecr. <b>14.0</b> -north-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.ecr. <b>15.0</b> -north-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.ecr. <b>16.0</b> -north-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.ecr. <b>17.0</b> -north-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
763104351884.dkr.ecr. <b>18.0</b> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.ecr. <b>19.0</b> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.ecr. <b>20.0</b> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.ecr. <b>21.0</b> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.ecr. <b>22.0</b> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.0.4</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.5</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.6</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.7</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.8</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.9</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.10</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.11</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.12</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.13</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.14</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.15</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.16</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.17</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.0.18</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.20</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.21</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.22</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.30</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.31</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.32</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.41</u> -north-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.edr. <u>14.1</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.edr. <u>14.0</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <u>15.0</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.edr. <u>15.2</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.0.3</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.e <u>0.0.4</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0.0.5</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0.0.6</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.0.7</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.0.8</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.0.9</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.0.10</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.0.11</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.0.12</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.0.13</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.0.14</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.0.15</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.0.16</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.0.17</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.13</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.20</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.21</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.22</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.30</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.31</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.32</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e <u>0.41</u> -north-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow', region='eu-north-1', version='1.0.0', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>0.10</u> -north-1.amazonaws.com/sagemaker-rl-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow:coach0.10-<tag>				
520713654638.dkr.e0.10.1 north-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.1 north-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.0 north-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.1 north-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inference (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow', region='eu-north-1', version='1.15.0', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
601324751636.dkr.e0.15.0 north-1.amazonaws.com/sagemaker-neo-tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow', region='eu-north-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.5eu-north-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.5.3eu-north-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6eu-north-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6.5eu-north-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5-<tag>		training	CPU, GPU	py3

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost',region='eu-north-1',version='1.2-1')
```

Registry path	Version	Job types (image scope)		
662702820516.dkr.ecr.eu-0.90-1north-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
662702820516.dkr.ecr.eu-0.90-2north-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
662702820516.dkr.ecr.eu-1.0-1north-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
662702820516.dkr.ecr.eu-1.2-1north-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		

Registry path	Version	Job types (image scope)		
662702820516.dkr.ecr.eu-1.2-2 north-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
662702820516.dkr.ecr.eu-1.3-1 north-1.amazonaws.com/ sagemaker- xgboost:<tag>		inference, training		
669576153137.dkr.ecr.eu-1 north-1.amazonaws.com/ xgboost:<tag>		inference, training		

### Docker Registry Paths and Example Code for Europe (Milan) (eu-south-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

#### Topics

- [BlazingText \(algorithm\) \(p. 1247\)](#)
- [Chainer \(DLC\) \(p. 1247\)](#)
- [Clarify \(algorithm\) \(p. 1248\)](#)
- [Data Wrangler \(algorithm\) \(p. 1248\)](#)
- [Debugger \(algorithm\) \(p. 1248\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 1248\)](#)
- [Factorization Machines \(algorithm\) \(p. 1249\)](#)
- [Hugging Face \(algorithm\) \(p. 1249\)](#)
- [IP Insights \(algorithm\) \(p. 1250\)](#)
- [Image classification \(algorithm\) \(p. 1250\)](#)
- [Inferentia MXNet \(DLC\) \(p. 1250\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 1251\)](#)
- [K-Means \(algorithm\) \(p. 1251\)](#)
- [KNN \(algorithm\) \(p. 1251\)](#)
- [Linear Learner \(algorithm\) \(p. 1251\)](#)
- [MXNet \(DLC\) \(p. 1252\)](#)
- [MXNet Coach \(DLC\) \(p. 1254\)](#)
- [Model Monitor \(algorithm\) \(p. 1255\)](#)
- [NTM \(algorithm\) \(p. 1255\)](#)
- [Neo Image Classification \(algorithm\) \(p. 1255\)](#)
- [Neo MXNet \(DLC\) \(p. 1256\)](#)
- [Neo PyTorch \(DLC\) \(p. 1256\)](#)
- [Neo Tensorflow \(DLC\) \(p. 1257\)](#)
- [Neo XGBoost \(algorithm\) \(p. 1257\)](#)
- [Object Detection \(algorithm\) \(p. 1257\)](#)
- [Object2Vec \(algorithm\) \(p. 1257\)](#)
- [PCA \(algorithm\) \(p. 1258\)](#)

- [PyTorch \(DLC\) \(p. 1258\)](#)
- [Random Cut Forest \(algorithm\) \(p. 1260\)](#)
- [Scikit-learn \(algorithm\) \(p. 1261\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 1261\)](#)
- [Seq2Seq \(algorithm\) \(p. 1261\)](#)
- [Spark \(algorithm\) \(p. 1261\)](#)
- [SparkML Serving \(algorithm\) \(p. 1262\)](#)
- [Tensorflow \(DLC\) \(p. 1262\)](#)
- [Tensorflow Coach \(DLC\) \(p. 1269\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 1270\)](#)
- [Tensorflow Ray \(DLC\) \(p. 1270\)](#)
- [XGBoost \(algorithm\) \(p. 1271\)](#)

### [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext',region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-south-1.amazonaws.com/blazingtext:<tag>		inference, training		

### [Chainer \(DLC\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer',region='eu-south-1',version='5.0.0',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
048378556238.dkr.ecr.eu-south-1.amazonaws.com/sagemaker-chainer:<tag>	4.10.0	inference, training	CPU, GPU	py2, py3
048378556238.dkr.ecr.eu-south-1.amazonaws.com/sagemaker-chainer:<tag>	4.11.0	inference, training	CPU, GPU	py2, py3
048378556238.dkr.ecr.eu-south-1.amazonaws.com/sagemaker-chainer:<tag>	5.0.0	inference, training	CPU, GPU	py2, py3

## Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify',region='eu-south-1',version='1.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
638885417683.dkr.ecr.eu-1.0.south-1.amazonaws.com/sagemaker-clarify-processing:<tag>		processing		

## Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler',region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
488287956546.dkr.ecr.eu-1.x.south-1.amazonaws.com/sagemaker-data-wrangler-container:<tag>		processing		

## Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger',region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
563282790590.dkr.ecr.eu-latest.south-1.amazonaws.com/sagemaker-debugger-rules:<tag>		debugger		

## DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='forecasting-deepar', region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-1.south-1.amazonaws.com/forecasting-deepar:<tag>		inference, training		

### Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines', region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-1.south-1.amazonaws.com/factorization-machines:<tag>		inference, training		

### Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='eu-south-1', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
692866216735.dkr.ecr.eu-4.4.2.south-1.amazonaws.com/huggingface-pytorch-training:<tag>		training		
692866216735.dkr.ecr.eu-4.5.0.south-1.amazonaws.com/huggingface-pytorch-training:<tag>		training		
692866216735.dkr.ecr.eu-4.4.2.south-1.amazonaws.com/huggingface-tensorflow-training:<tag>		training		
692866216735.dkr.ecr.eu-4.5.0.south-1.amazonaws.com/		training		

Registry path	Version	Job types (image scope)		
huggingface-tensorflow-training:<tag>				

## IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights',region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-south-1.amazonaws.com/ipinsights:<tag>		inference, training		

## Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification',region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-south-1.amazonaws.com/image-classification:<tag>		inference, training		

## Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='eu-south-1',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
966458181534.dkr.ecr.eu-south-1.amazonaws.com/sagemaker-neo-mxnet:<tag>		inference	inf	py3

## Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='eu-south-1',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
966458181534.dkr.ecr.eu-south-1.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

## K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans',region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-south-1.amazonaws.com/kmeans:<tag>		inference, training		

## KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn',region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-south-1.amazonaws.com/knn:<tag>		inference, training		

## Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner',region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-south-1.amazonaws.com/linear-learner:<tag>		inference, training		

## MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='eu-south-1', version='1.4.1', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
048378556238.dkr.edr.10-south-1.amazonaws.com/sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3
048378556238.dkr.edr.10-south-1.amazonaws.com/sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
048378556238.dkr.edr.10-south-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
048378556238.dkr.edr.11-south-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2
048378556238.dkr.e0.12.1south-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
048378556238.dkr.e0.12.1south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
048378556238.dkr.e0.10-south-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
048378556238.dkr.e0.10-south-1.amazonaws.com/		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-mxnet:<tag>				
048378556238.dkr.edr.10-south-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
048378556238.dkr.edr.10-south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
048378556238.dkr.edr.21-south-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
048378556238.dkr.edr.21-south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
048378556238.dkr.edr.30-south-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
048378556238.dkr.edr.30-south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
048378556238.dkr.edr.40-south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
048378556238.dkr.edr.41-south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
692866216735.dkr.edr.41-south-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
692866216735.dkr.edr.51-south-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
692866216735.dkr.eur.2.0-south-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py3
692866216735.dkr.eur.4.1-south-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
692866216735.dkr.eur.6.0-south-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
692866216735.dkr.eur.7.0-south-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
692866216735.dkr.eur.8.0-south-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37
692866216735.dkr.eur.4.1-south-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
692866216735.dkr.eur.6.0-south-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
692866216735.dkr.eur.7.0-south-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
692866216735.dkr.eur.8.0-south-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

## MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='eu-south-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
048378556238.dkr.ecr.eu-south-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
048378556238.dkr.ecr.eu-south-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor', region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
933208885752.dkr.ecr.eu-south-1.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm', region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-south-1.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo', region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
966458181534.dkr.ecr.eu-south-1.amazonaws.com/image-classification-neo:<tag>	latest	inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet', region='eu-south-1', version='1.8', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
966458181534.dkr.ecr.eu-south-1.amazonaws.com/sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='eu-south-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
966458181534.dkr.ecr.eu-south-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
966458181534.dkr.ecr.eu-south-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
966458181534.dkr.ecr.eu-south-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

## Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow',region='eu-south-1',version='1.15.3',py_version='py3',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
966458181534.dkr.ecr.eu-south-1.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

## Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo',region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
966458181534.dkr.ecr.eu-south-1.amazonaws.com/xgboost-neo:<tag>	latest	inference		

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection',region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-south-1.amazonaws.com/object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec',region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-1.south-1.amazonaws.com/object2vec:<tag>		inference, training		

### PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca', region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-1.south-1.amazonaws.com/pca:<tag>		inference, training		

### PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch', region='eu-south-1', version='1.8.0', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
048378556238.dkr.e0.40-south-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
048378556238.dkr.e0.40-south-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
048378556238.dkr.edr.0.0-south-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
048378556238.dkr.edr.0.0-south-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
048378556238.dkr.edr. <del>10</del> -south-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
048378556238.dkr.edr. <del>10</del> -south-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
692866216735.dkr.edr. <del>11</del> -south-1.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
692866216735.dkr.edr. <del>20</del> -south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
692866216735.dkr.edr. <del>31</del> -south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
692866216735.dkr.edr. <del>40</del> -south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
692866216735.dkr.edr. <del>50</del> -south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
692866216735.dkr.edr. <del>60</del> -south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
692866216735.dkr.edr. <del>71</del> -south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
692866216735.dkr.edr. <del>80</del> -south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
692866216735.dkr.edr. <del>81</del> -south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36

Registry path	Version	Job types (image scope)	Processor types	Python versions
692866216735.dkr.ecr. <b>eu-1</b> -south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
692866216735.dkr.ecr. <b>eu-1</b> -south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
692866216735.dkr.ecr. <b>eu-1</b> -south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
692866216735.dkr.ecr. <b>eu-1</b> -south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
692866216735.dkr.ecr. <b>eu-1</b> -south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
692866216735.dkr.ecr. <b>eu-1</b> -south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
692866216735.dkr.ecr. <b>eu-1</b> -south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
692866216735.dkr.ecr. <b>eu-1</b> -south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36

## Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-1-south-1.amazonaws.com/randomcutforest:<tag>		inference, training		

## Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn',region='eu-south-1',version='0.23-1',image_scope='inference')
```

Registry path	Version	Job types (image scope)		
978288397137.dkr.ecr.eu-0.20.0.south-1.amazonaws.com/sagemaker-scikit-learn:<tag>		inference, training		
978288397137.dkr.ecr.eu-0.23-1.south-1.amazonaws.com/sagemaker-scikit-learn:<tag>		inference, training		

## Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-1.south-1.amazonaws.com/semantic-segmentation:<tag>		inference, training		

## Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='eu-south-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-1.south-1.amazonaws.com/seq2seq:<tag>		inference, training		

## Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='eu-south-1',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
753923664805.dkr.ecr.eu-2.4.south-1.amazonaws.com/sagemaker-spark-processing:<tag>		processing		
753923664805.dkr.ecr.eu-3.0.south-1.amazonaws.com/sagemaker-spark-processing:<tag>		processing		

### SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving',region='eu-south-1',version='2.4')
```

Registry path	Version	Job types (image scope)		
978288397137.dkr.ecr.eu-2.2.south-1.amazonaws.com/sagemaker-sparkml-serving:<tag>		inference		
978288397137.dkr.ecr.eu-2.4.south-1.amazonaws.com/sagemaker-sparkml-serving:<tag>		inference		

### Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow',region='eu-south-1',version='1.12.0',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
048378556238.dkr.ecr.eu-10.0.south-1.amazonaws.com/sagemaker-tensorflow-eia:<tag>		eia	CPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
048378556238.dkr.e <u>11</u> :0 south-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
048378556238.dkr.e <u>12</u> :0 south-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
048378556238.dkr.e <u>13</u> :1 south-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
048378556238.dkr.e <u>14</u> :0 south-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
048378556238.dkr.e <u>15</u> :0 south-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
048378556238.dkr.e <u>16</u> :0 south-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
048378556238.dkr.e <u>17</u> :0 south-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
048378556238.dkr.e <u>18</u> :0 south-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
048378556238.dkr.e <u>19</u> :0 south-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
048378556238.dkr.edr. <b>10.0</b> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
048378556238.dkr.edr. <b>11</b> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
048378556238.dkr.edr. <b>11</b> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
048378556238.dkr.edr. <b>5.0</b> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
048378556238.dkr.edr. <b>5.0</b> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
048378556238.dkr.edr. <b>6.0</b> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
048378556238.dkr.edr. <b>6.0</b> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
048378556238.dkr.edr. <b>7.0</b> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
048378556238.dkr.edr. <b>7.0</b> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
048378556238.dkr.edr. <b>8.0</b> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
048378556238.dkr.edr. <b>8.0</b> -south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
048378556238.dkr.e <u>9</u> .0-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
048378556238.dkr.e <u>9</u> .0-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
692866216735.dkr.e <u>14</u> .0-south-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
692866216735.dkr.e <u>15</u> .0-south-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
692866216735.dkr.e <u>20</u> .0-south-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
692866216735.dkr.e <u>21</u> .0-south-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
692866216735.dkr.e <u>25</u> .0-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>24</u> .0-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>25</u> .0-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>25</u> .2-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
692866216735.dkr.e <u>0.1</u> 3-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.1</u> 4-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.1</u> 5-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.1</u> 0-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.1</u> 1-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.1</u> 2-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.1</u> 3-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.1</u> 4-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.1</u> 0-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.1</u> 1-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.1</u> 2-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
692866216735.dkr.e <u>0.13</u> -south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.20</u> -south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.21</u> -south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.22</u> -south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.30</u> -south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.31</u> -south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.32</u> -south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.41</u> -south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
692866216735.dkr.e <u>0.13</u> -1south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
692866216735.dkr.e <u>0.14</u> .0south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
692866216735.dkr.e <u>0.15</u> .0south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
692866216735.dkr.e <u>15</u> .2 south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
692866216735.dkr.e <u>15</u> .3 south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
692866216735.dkr.e <u>15</u> .4 south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
692866216735.dkr.e <u>15</u> .5 south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
692866216735.dkr.e <u>0</u> . <u>0</u> - south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
692866216735.dkr.e <u>0</u> . <u>0</u> 1- south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
692866216735.dkr.e <u>0</u> . <u>0</u> 2- south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
692866216735.dkr.e <u>0</u> . <u>0</u> 3- south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
692866216735.dkr.e <u>0</u> . <u>0</u> 4- south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
692866216735.dkr.e <u>0</u> . <u>1</u> 0- south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
692866216735.dkr.e <u>0</u> . <u>1</u> 1- south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
692866216735.dkr.e <u>0.10</u> -south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
692866216735.dkr.e <u>0.15</u> -south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
692866216735.dkr.e <u>0.20</u> -south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
692866216735.dkr.e <u>0.21</u> -south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
692866216735.dkr.e <u>0.22</u> -south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
692866216735.dkr.e <u>0.30</u> -south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
692866216735.dkr.e <u>0.31</u> -south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
692866216735.dkr.e <u>0.32</u> -south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
692866216735.dkr.e <u>0.41</u> -south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

## Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow', region='eu-south-1', version='1.0.0', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
048378556238.dkr.e0.10-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
048378556238.dkr.e0.10.1-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
048378556238.dkr.e0.11-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
048378556238.dkr.e0.11.0-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
048378556238.dkr.e0.11.1-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

### Tensorflow Inferentia (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow', region='eu-south-1', version='1.15.0', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
966458181534.dkr.e0.15.0-south-1.amazonaws.com/sagemaker-neo-tensorflow:<tag>		inference	inf	py3

### Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='ray-tensorflow',region='eu-south-1',version='0.8.5',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
048378556238.dkr.e <u>0.5</u> -south-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5-<tag>		training	CPU, GPU	py3
048378556238.dkr.e <u>0.5.3</u> -south-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
048378556238.dkr.e <u>0.6</u> -south-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6-<tag>		training	CPU, GPU	py3
048378556238.dkr.e <u>0.6.5</u> -south-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5-<tag>		training	CPU, GPU	py3

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost',region='eu-south-1',version='1.2-1')
```

Registry path	Version	Job types (image scope)		
257386234256.dkr.ecr.eu-1-south-1.amazonaws.com/xgboost:<tag>		inference, training		
978288397137.dkr.ecr.eu-0.90-1-south-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
978288397137.dkr.ecr.eu-0.90-2-south-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
978288397137.dkr.ecr.eu-1.0-1-south-1.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
sagemaker-xgboost:<tag>				
978288397137.dkr.ecr.eu-1.2-1 south-1.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
978288397137.dkr.ecr.eu-1.2-2 south-1.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		
978288397137.dkr.ecr.eu-1.3-1 south-1.amazonaws.com/ sagemaker-xgboost:<tag>		inference, training		

## Docker Registry Paths and Example Code for Middle East (Bahrain) (me-south-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

### Topics

- [BlazingText \(algorithm\) \(p. 1273\)](#)
- [Chainer \(DLC\) \(p. 1273\)](#)
- [Clarify \(algorithm\) \(p. 1274\)](#)
- [Data Wrangler \(algorithm\) \(p. 1274\)](#)
- [Debugger \(algorithm\) \(p. 1274\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 1275\)](#)
- [Factorization Machines \(algorithm\) \(p. 1275\)](#)
- [Hugging Face \(algorithm\) \(p. 1275\)](#)
- [IP Insights \(algorithm\) \(p. 1276\)](#)
- [Image classification \(algorithm\) \(p. 1276\)](#)
- [Inferentia MXNet \(DLC\) \(p. 1276\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 1277\)](#)
- [K-Means \(algorithm\) \(p. 1277\)](#)
- [KNN \(algorithm\) \(p. 1277\)](#)
- [Linear Learner \(algorithm\) \(p. 1278\)](#)
- [MXNet \(DLC\) \(p. 1278\)](#)
- [MXNet Coach \(DLC\) \(p. 1281\)](#)
- [Model Monitor \(algorithm\) \(p. 1281\)](#)
- [NTM \(algorithm\) \(p. 1281\)](#)
- [Neo Image Classification \(algorithm\) \(p. 1282\)](#)
- [Neo MXNet \(DLC\) \(p. 1282\)](#)
- [Neo PyTorch \(DLC\) \(p. 1282\)](#)
- [Neo Tensorflow \(DLC\) \(p. 1283\)](#)
- [Neo XGBoost \(algorithm\) \(p. 1283\)](#)

- [Object Detection \(algorithm\) \(p. 1284\)](#)
- [Object2Vec \(algorithm\) \(p. 1284\)](#)
- [PCA \(algorithm\) \(p. 1284\)](#)
- [PyTorch \(DLC\) \(p. 1284\)](#)
- [Random Cut Forest \(algorithm\) \(p. 1287\)](#)
- [Scikit-learn \(algorithm\) \(p. 1287\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 1287\)](#)
- [Seq2Seq \(algorithm\) \(p. 1288\)](#)
- [Spark \(algorithm\) \(p. 1288\)](#)
- [SparkML Serving \(algorithm\) \(p. 1288\)](#)
- [Tensorflow \(DLC\) \(p. 1289\)](#)
- [Tensorflow Coach \(DLC\) \(p. 1296\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 1297\)](#)
- [Tensorflow Ray \(DLC\) \(p. 1297\)](#)
- [XGBoost \(algorithm\) \(p. 1298\)](#)

### [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext', region='me-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.me-south-1.amazonaws.com/blazingtext:<tag>		inference, training		

### [Chainer \(DLC\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer', region='me-south-1', version='5.0.0', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
724002660598.dkr.ecr.me-south-1.amazonaws.com/sagemaker-chainer:<tag>	4.0.0-	inference, training	CPU, GPU	py2, py3
724002660598.dkr.ecr.me-south-1.amazonaws.com/sagemaker-chainer:<tag>	4.1.0-	inference, training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
724002660598.dkr.ecr.me1.0-south-1.amazonaws.com/sagemaker-chainer:<tag>		inference, training	CPU, GPU	py2, py3

### Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify', region='me-south-1', version='1.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
835444307964.dkr.ecr.me1.0-south-1.amazonaws.com/sagemaker-clarify-processing:<tag>		processing		

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler', region='me-south-1')
```

Registry path	Version	Job types (image scope)		
376037874950.dkr.ecr.me1.x-south-1.amazonaws.com/sagemaker-data-wrangler-container:<tag>		processing		

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger', region='me-south-1')
```

Registry path	Version	Job types (image scope)		
986000313247.dkr.ecr.me1atest-south-1.amazonaws.com/		debugger		

Registry path	Version	Job types (image scope)		
sagemaker-debugger-rules:<tag>				

## DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar', region='us-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.us-south-1.amazonaws.com/forecasting-deepar:<tag>		inference, training		

## Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines', region='us-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.us-south-1.amazonaws.com/factorization-machines:<tag>		inference, training		

## Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='us-south-1', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
217643126080.dkr.ecr.us-south-1.amazonaws.com/huggingface-pytorch-training:<tag>	4.4.2	training		

Registry path	Version	Job types (image scope)		
217643126080.dkr.ecr.me4.5.0 south-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
217643126080.dkr.ecr.me4.4.2 south-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		
217643126080.dkr.ecr.me4.5.0 south-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		

### IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights', region='me-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.me4 south-1.amazonaws.com/ ipinsights:<tag>		inference, training		

### Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification', region='me-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.me4 south-1.amazonaws.com/ image- classification:<tag>		inference, training		

### Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet',region='me-south-1',version='1.5.1',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
836785723513.dkr.ecr.me-south-1.amazonaws.com/sagemaker-neo-mxnet:<tag>		inference	inf	py3

### Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch',region='me-south-1',version='1.5.1',py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
836785723513.dkr.ecr.me-south-1.amazonaws.com/sagemaker-neo-pytorch:<tag>		inference	inf	py3

### K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans',region='me-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.me-south-1.amazonaws.com/kmeans:<tag>		inference, training		

### KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn',region='me-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.me1 south-1.amazonaws.com/ knn:<tag>		inference, training		

### Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='me-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.me1 south-1.amazonaws.com/ linear-learner:<tag>		inference, training		

### MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='me-south-1', version='1.4.1', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
217643126080.dkr.ecr.me1-south-1.amazonaws.com/ mxnet-inference-eia:<tag>		eia	CPU	py2, py3
217643126080.dkr.ecr.me1-south-1.amazonaws.com/ mxnet-inference-eia:<tag>		eia	CPU	py2, py3
217643126080.dkr.ecr.me1-south-1.amazonaws.com/ mxnet-inference-eia:<tag>		eia	CPU	py3
217643126080.dkr.ecr.me1-south-1.amazonaws.com/ mxnet-inference:<tag>		inference	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
217643126080.dkr.edr. <del>6.0</del> -south-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
217643126080.dkr.edr. <del>7.0</del> -south-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
217643126080.dkr.edr. <del>8.0</del> -south-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37
217643126080.dkr.edr. <del>4.1</del> -south-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
217643126080.dkr.edr. <del>6.0</del> -south-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
217643126080.dkr.edr. <del>7.0</del> -south-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
217643126080.dkr.edr. <del>8.0</del> -south-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37
724002660598.dkr.edr. <del>3.0</del> -south-1.amazonaws.com/sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3
724002660598.dkr.edr. <del>4.0</del> -south-1.amazonaws.com/sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
724002660598.dkr.edr. <del>4.0</del> -south-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
724002660598.dkr.edr. <del>4.1</del> -south-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
724002660598.dkr.edr.12e1-south-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
724002660598.dkr.edr.12e1-south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
724002660598.dkr.edr.01e-south-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
724002660598.dkr.edr.01e-south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
724002660598.dkr.edr.11e-south-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
724002660598.dkr.edr.11e-south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
724002660598.dkr.edr.21e-south-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
724002660598.dkr.edr.21e-south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
724002660598.dkr.edr.31e-south-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
724002660598.dkr.edr.31e-south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
724002660598.dkr.edr.41e-south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
724002660598.dkr.ecr.me-south-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2

### MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='me-south-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
724002660598.dkr.ecr.me-south-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
724002660598.dkr.ecr.me-south-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

### Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor', region='me-south-1')
```

Registry path	Version	Job types (image scope)		
607024016150.dkr.ecr.me-south-1.amazonaws.com/sagemaker-model-monitor-analyzer:<tag>		monitoring		

### NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='ntm', region='me-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.me-south-1.amazonaws.com/ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo', region='me-south-1')
```

Registry path	Version	Job types (image scope)		
836785723513.dkr.ecr.me-south-1.amazonaws.com/image-classification-neo:<tag>		inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet', region='me-south-1', version='1.8', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
836785723513.dkr.ecr.me-south-1.amazonaws.com/sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='me-south-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
836785723513.dkr.ecr. <del>me-</del> south-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
836785723513.dkr.ecr. <del>me-</del> south-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
836785723513.dkr.ecr. <del>me-</del> south-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

### Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='me-south-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
836785723513.dkr.ecr. <del>me-</del> south-1.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

### Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='me-south-1')
```

Registry path	Version	Job types (image scope)		
836785723513.dkr.ecr.melatest.south-1.amazonaws.com/xgboost-neo:<tag>		inference		

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection',region='me-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.me1 south-1.amazonaws.com/ object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec',region='me-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.me1 south-1.amazonaws.com/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca',region='me-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.me1 south-1.amazonaws.com/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch',region='me-
south-1',version='1.8.0',py_version='py3',image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
217643126080.dkr.edr.31e-south-1.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
217643126080.dkr.edr.21e-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
217643126080.dkr.edr.31e-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
217643126080.dkr.edr.41e-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
217643126080.dkr.edr.51e-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
217643126080.dkr.edr.61e-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
217643126080.dkr.edr.71e-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
217643126080.dkr.edr.81e-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
217643126080.dkr.edr.81b-south-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
217643126080.dkr.edr.21e-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
217643126080.dkr.edr.31e-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
217643126080.dkr.edr.40-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
217643126080.dkr.edr.50-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3
217643126080.dkr.edr.60-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
217643126080.dkr.edr.70-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
217643126080.dkr.edr.80-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
217643126080.dkr.edr.80e-south-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py3, py36
724002660598.dkr.edr.40-south-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
724002660598.dkr.edr.40-south-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
724002660598.dkr.edr.00e-south-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
724002660598.dkr.edr.00e-south-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
724002660598.dkr.edr.10-south-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
724002660598.dkr.ecr.me-south-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3

### Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest',region='me-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.me-south-1.amazonaws.com/randomcutforest:<tag>		inference, training		

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn',region='me-south-1',version='0.23-1',image_scope='inference')
```

Registry path	Version	Job types (image scope)		
801668240914.dkr.ecr.me0.20.0-south-1.amazonaws.com/sagemaker-scikit-learn:<tag>		inference, training		
801668240914.dkr.ecr.me0.23-1-south-1.amazonaws.com/sagemaker-scikit-learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='me-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.me1 south-1.amazonaws.com/ semantic-segmentation:<tag>		inference, training		

## Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='me-south-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.me1 south-1.amazonaws.com/ seq2seq:<tag>		inference, training		

## Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='me-south-1',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
750251592176.dkr.ecr.me2.4 south-1.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		
750251592176.dkr.ecr.me3.0 south-1.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving',region='me-south-1',version='2.4')
```

Registry path	Version	Job types (image scope)		
801668240914.dkr.ecr.me2.2 south-1.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		
801668240914.dkr.ecr.me2.4 south-1.amazonaws.com/ sagemaker-sparkml- serving:<tag>		inference		

### Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='me-'
south-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
217643126080.dkr.eifr.14e0 south-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
217643126080.dkr.eifr.15e0 south-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
217643126080.dkr.eifr.20e0- south-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
217643126080.dkr.eifr.20e0- south-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
217643126080.dkr.eifr.13e0 south-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
217643126080.dkr.eifr.14e0 south-1.amazonaws.com/		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow-inference:<tag>				
217643126080.dkr.e0r.15e0-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
217643126080.dkr.e0r.15e2-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
217643126080.dkr.e0r.15e3-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
217643126080.dkr.e0r.15e4-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
217643126080.dkr.e0r.15e5-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
217643126080.dkr.e0r.00e-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
217643126080.dkr.e0r.00e-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
217643126080.dkr.e0r.00e-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
217643126080.dkr.e0r.00e-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
217643126080.dkr.e0r.00e-south-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
217643126080.dkr.e2.1m-south-1.amazonaws.com/tensorflow-inference:<tag>	e2.1m-	inference	CPU, GPU	-
217643126080.dkr.e2.1m-south-1.amazonaws.com/tensorflow-inference:<tag>	e2.1m-	inference	CPU, GPU	-
217643126080.dkr.e2.1m-south-1.amazonaws.com/tensorflow-inference:<tag>	e2.1m-	inference	CPU, GPU	-
217643126080.dkr.e2.1m-south-1.amazonaws.com/tensorflow-inference:<tag>	e2.1m-	inference	CPU, GPU	-
217643126080.dkr.e2.2m-south-1.amazonaws.com/tensorflow-inference:<tag>	e2.2m-	inference	CPU, GPU	-
217643126080.dkr.e2.2m-south-1.amazonaws.com/tensorflow-inference:<tag>	e2.2m-	inference	CPU, GPU	-
217643126080.dkr.e2.2m-south-1.amazonaws.com/tensorflow-inference:<tag>	e2.2m-	inference	CPU, GPU	-
217643126080.dkr.e2.3m-south-1.amazonaws.com/tensorflow-inference:<tag>	e2.3m-	inference	CPU, GPU	-
217643126080.dkr.e2.3m-south-1.amazonaws.com/tensorflow-inference:<tag>	e2.3m-	inference	CPU, GPU	-
217643126080.dkr.e2.3m-south-1.amazonaws.com/tensorflow-inference:<tag>	e2.3m-	inference	CPU, GPU	-
217643126080.dkr.e2.4m-south-1.amazonaws.com/tensorflow-inference:<tag>	e2.4m-	inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
217643126080.dkr.e <u>13e1</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
217643126080.dkr.e <u>14e0</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
217643126080.dkr.e <u>15e0</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
217643126080.dkr.e <u>15e2</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
217643126080.dkr.e <u>15e3</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
217643126080.dkr.e <u>15e4</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
217643126080.dkr.e <u>15e5</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
217643126080.dkr.e <u>2.0e-</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
217643126080.dkr.e <u>2.0e1-</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
217643126080.dkr.e <u>2.0e2-</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
217643126080.dkr.e <u>2.0e3-</u> south-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
217643126080.dkr.e2.01e-south-1.amazonaws.com/tensorflow-training:<tag>	e2.01e-	training	CPU, GPU	py3
217643126080.dkr.e2.11e-south-1.amazonaws.com/tensorflow-training:<tag>	e2.11e-	training	CPU, GPU	py2, py3
217643126080.dkr.e2.11e-south-1.amazonaws.com/tensorflow-training:<tag>	e2.11e-	training	CPU, GPU	py2, py3
217643126080.dkr.e2.11e-south-1.amazonaws.com/tensorflow-training:<tag>	e2.11e-	training	CPU, GPU	py3
217643126080.dkr.e2.11e-south-1.amazonaws.com/tensorflow-training:<tag>	e2.11e-	training	CPU, GPU	py3
217643126080.dkr.e2.20e-south-1.amazonaws.com/tensorflow-training:<tag>	e2.20e-	training	CPU, GPU	py37
217643126080.dkr.e2.21e-south-1.amazonaws.com/tensorflow-training:<tag>	e2.21e-	training	CPU, GPU	py37
217643126080.dkr.e2.21e-south-1.amazonaws.com/tensorflow-training:<tag>	e2.21e-	training	CPU, GPU	py37
217643126080.dkr.e2.30e-south-1.amazonaws.com/tensorflow-training:<tag>	e2.30e-	training	CPU, GPU	py37
217643126080.dkr.e2.31e-south-1.amazonaws.com/tensorflow-training:<tag>	e2.31e-	training	CPU, GPU	py37
217643126080.dkr.e2.31e-south-1.amazonaws.com/tensorflow-training:<tag>	e2.31e-	training	CPU, GPU	py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
217643126080.dkr.e0.41e-south-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
724002660598.dkr.e0.10e0south-1.amazonaws.com/sagemaker-tensorflow-eia:<tag>		eia	CPU	py2
724002660598.dkr.e0.11e0south-1.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2, py3
724002660598.dkr.e0.12e0south-1.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2, py3
724002660598.dkr.e0.13e1south-1.amazonaws.com/sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2
724002660598.dkr.e0.11e0south-1.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>		eia	CPU	-
724002660598.dkr.e0.12e0south-1.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>		eia	CPU	-
724002660598.dkr.e0.13e0south-1.amazonaws.com/sagemaker-tensorflow-serving-eia:<tag>		eia	CPU	-
724002660598.dkr.e0.11e0south-1.amazonaws.com/sagemaker-tensorflow-serving:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
724002660598.dkr.edr.12e0-south-1.amazonaws.com/sagemaker-tensorflow-serving:<tag>		inference	CPU, GPU	-
724002660598.dkr.edr.10e0-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
724002660598.dkr.edr.10e0-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
724002660598.dkr.edr.41e-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
724002660598.dkr.edr.41e-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
724002660598.dkr.edr.51e-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
724002660598.dkr.edr.51e-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
724002660598.dkr.edr.61e-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
724002660598.dkr.edr.61e-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
724002660598.dkr.edr.71e-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
724002660598.dkr.edr.71e-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
724002660598.dkr.e0.80e-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
724002660598.dkr.e0.80e-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
724002660598.dkr.e0.90e-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
724002660598.dkr.e0.90e-south-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow', region='me-south-1', version='1.0.0', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
724002660598.dkr.e0.10e-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
724002660598.dkr.e0.10e1south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
724002660598.dkr.e0.11e-south-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
724002660598.dkr.e0.11e0south-1.amazonaws.com/sagemaker-rl-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow:coach0.11.0-<tag>				
724002660598.dkr.e0.11e1 south-1.amazonaws.com/ sagemaker-rl- tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

### Tensorflow Inference (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow', region='me-south-1', version='1.15.0', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
836785723513.dkr.e0.15e0 south-1.amazonaws.com/ sagemaker-neo- tensorflow:<tag>		inference	inf	py3

### Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow', region='me-south-1', version='0.8.5', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
724002660598.dkr.e0.5ne- south-1.amazonaws.com/ sagemaker-rl- tensorflow:ray0.5-<tag>		training	CPU, GPU	py3
724002660598.dkr.e0.5n1- south-1.amazonaws.com/ sagemaker-rl- tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
724002660598.dkr.e0.5ne- south-1.amazonaws.com/		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-rl-tensorflow:ray0.6-<tag>				
724002660598.dkr.ecr.me0.6.5-south-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5-<tag>	0.6.5	training	CPU, GPU	py3

### XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost', region='me-south-1', version='1.2-1')
```

Registry path	Version	Job types (image scope)		
249704162688.dkr.ecr.me1-south-1.amazonaws.com/xgboost:<tag>		inference, training		
801668240914.dkr.ecr.me0.90-1south-1.amazonaws.com/sagemaker-xgboost:<tag>	0.90-1	inference, training		
801668240914.dkr.ecr.me0.90-2south-1.amazonaws.com/sagemaker-xgboost:<tag>	0.90-2	inference, training		
801668240914.dkr.ecr.me1.0-1south-1.amazonaws.com/sagemaker-xgboost:<tag>	1.0-1	inference, training		
801668240914.dkr.ecr.me1.2-1south-1.amazonaws.com/sagemaker-xgboost:<tag>	1.2-1	inference, training		
801668240914.dkr.ecr.me1.2-2south-1.amazonaws.com/sagemaker-xgboost:<tag>	1.2-2	inference, training		
801668240914.dkr.ecr.me1.3-1south-1.amazonaws.com/sagemaker-xgboost:<tag>	1.3-1	inference, training		

## Docker Registry Paths and Example Code for South America (São Paulo) (sa-east-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

### Topics

- [BlazingText \(algorithm\) \(p. 1300\)](#)
- [Chainer \(DLC\) \(p. 1300\)](#)
- [Clarify \(algorithm\) \(p. 1300\)](#)
- [Data Wrangler \(algorithm\) \(p. 1301\)](#)
- [Debugger \(algorithm\) \(p. 1301\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 1301\)](#)
- [Factorization Machines \(algorithm\) \(p. 1302\)](#)
- [Hugging Face \(algorithm\) \(p. 1302\)](#)
- [IP Insights \(algorithm\) \(p. 1302\)](#)
- [Image classification \(algorithm\) \(p. 1303\)](#)
- [Inferentia MXNet \(DLC\) \(p. 1303\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 1303\)](#)
- [K-Means \(algorithm\) \(p. 1304\)](#)
- [KNN \(algorithm\) \(p. 1304\)](#)
- [Linear Learner \(algorithm\) \(p. 1304\)](#)
- [MXNet \(DLC\) \(p. 1304\)](#)
- [MXNet Coach \(DLC\) \(p. 1307\)](#)
- [Model Monitor \(algorithm\) \(p. 1308\)](#)
- [NTM \(algorithm\) \(p. 1308\)](#)
- [Neo Image Classification \(algorithm\) \(p. 1308\)](#)
- [Neo MXNet \(DLC\) \(p. 1308\)](#)
- [Neo PyTorch \(DLC\) \(p. 1309\)](#)
- [Neo Tensorflow \(DLC\) \(p. 1309\)](#)
- [Neo XGBoost \(algorithm\) \(p. 1310\)](#)
- [Object Detection \(algorithm\) \(p. 1310\)](#)
- [Object2Vec \(algorithm\) \(p. 1310\)](#)
- [PCA \(algorithm\) \(p. 1311\)](#)
- [PyTorch \(DLC\) \(p. 1311\)](#)
- [Random Cut Forest \(algorithm\) \(p. 1313\)](#)
- [Scikit-learn \(algorithm\) \(p. 1313\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 1314\)](#)
- [Seq2Seq \(algorithm\) \(p. 1314\)](#)
- [Spark \(algorithm\) \(p. 1314\)](#)
- [SparkML Serving \(algorithm\) \(p. 1315\)](#)
- [Tensorflow \(DLC\) \(p. 1315\)](#)
- [Tensorflow Coach \(DLC\) \(p. 1322\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 1323\)](#)
- [Tensorflow Ray \(DLC\) \(p. 1323\)](#)
- [XGBoost \(algorithm\) \(p. 1324\)](#)

## BlazingText (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext',region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-1 east-1.amazonaws.com/ blazingtext:<tag>		inference, training		

## Chainer (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer',region='sa-
east-1',version='5.0.0',py_version='py3',image_scope='inference',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e4.60 east-1.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.e4.10 east-1.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3
520713654638.dkr.e5.60 east-1.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3

## Clarify (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='clarify',region='sa-
east-1',version='1.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
520018980103.dkr.ecr.sa-1.0 east-1.amazonaws.com/		processing		

Registry path	Version	Job types (image scope)		
sagemaker-clarify-processing:<tag>				

### Data Wrangler (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='data-wrangler', region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
424196993095.dkr.ecr.sa-east-1.x east-1.amazonaws.com/ sagemaker- data-wrangler- container:<tag>		processing		

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger', region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
818342061345.dkr.ecr.sa-latest east-1.amazonaws.com/ sagemaker-debugger- rules:<tag>		debugger		

### DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar', region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-1 east-1.amazonaws.com/ forecasting- deepar:<tag>		inference, training		

## Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines',region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-1 east-1.amazonaws.com/ factorization- machines:<tag>		inference, training		

## Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface',region='sa-  
east-1',version='4.4.2',image_scope='training',base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
763104351884.dkr.ecr.sa-4.4.2 east-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.sa-4.5.0 east-1.amazonaws.com/ huggingface-pytorch- training:<tag>		training		
763104351884.dkr.ecr.sa-4.4.2 east-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		
763104351884.dkr.ecr.sa-4.5.0 east-1.amazonaws.com/ huggingface- tensorflow- training:<tag>		training		

## IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights',region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-1 east-1.amazonaws.com/ ipinsights:<tag>		inference, training		

### Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification', region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-1 east-1.amazonaws.com/ image- classification:<tag>		inference, training		

### Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet', region='sa-  
east-1', version='1.5.1', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
756306329178.dkr.ecr.sa-1 east-1.amazonaws.com/ sagemaker-neo- mxnet:<tag>		inference	inf	py3

### Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch', region='sa-  
east-1', version='1.5.1', py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
756306329178.dkr.ecr.sa-1 east-1.amazonaws.com/		inference	inf	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-neo-pytorch:<tag>				

## K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans',region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-1 east-1.amazonaws.com/ kmeans:<tag>		inference, training		

## KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn',region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-1 east-1.amazonaws.com/ knn:<tag>		inference, training		

## Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner',region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-1 east-1.amazonaws.com/ linear-learner:<tag>		inference, training		

## MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='sa-east-1', version='1.4.1', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e-dr.3@east-1.amazonaws.com/sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.e-dr.4@east-1.amazonaws.com/sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
520713654638.dkr.e-dr.4@east-1.amazonaws.com/sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.12.1-east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e0.12.1-east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.12.1-east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e-dr.6@east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e-dr.6@east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e-dr.1@east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e-dr.1@east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e-dr.2@-east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e-dr.2@-east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e-dr.3@-east-1.amazonaws.com/sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e-dr.3@-east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e-dr.4@-east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e-dr.4@-east-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
763104351884.dkr.e-dr.4@-east-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.e-dr.5@-east-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
763104351884.dkr.e-dr.5@-east-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.e-dr.6@-east-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.e-dr.6@-east-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e0r.7a0 east-1.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.e0r.8a0 east-1.amazonaws.com/ mxnet- inference:<tag>		inference	CPU, GPU	py37
763104351884.dkr.e0r.4a1 east-1.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e0r.6a0 east-1.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e0r.7a0 east-1.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e0r.8a0 east-1.amazonaws.com/ mxnet- training:<tag>		training	CPU, GPU	py37

## MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='sa-
east-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0r.3a0- east-1.amazonaws.com/ sagemaker-rl- mxnet:coach0.11- <tag>		training	CPU, GPU	py3
520713654638.dkr.e0r.3a0- east-1.amazonaws.com/ sagemaker-rl- mxnet:coach0.11.0- <tag>		training	CPU, GPU	py3

## Model Monitor (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='model-monitor',region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
539772159869.dkr.ecr.sa-east-1.amazonaws.com/ sagemaker-model-monitor-analyzer:<tag>		monitoring		

## NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm',region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-1 east-1.amazonaws.com/ ntm:<tag>		inference, training		

## Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo',region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
756306329178.dkr.ecr.sa-latest east-1.amazonaws.com/ image-classification-neo:<tag>		inference		

## Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
```

```
image_uris.retrieve(framework='neo-mxnet', region='sa-east-1', version='1.8', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
756306329178.dkr.e-dr.8a-east-1.amazonaws.com/sagemaker-inference-mxnet:<tag>		inference	CPU, GPU	py3

## Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='sa-east-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
756306329178.dkr.e-dr.4a-east-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
756306329178.dkr.e-dr.5a-east-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
756306329178.dkr.e-dr.6a-east-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

## Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='sa-east-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
756306329178.dkr.ecr.sa-1 east-1.amazonaws.com/ sagemaker- inference- tensorflow:<tag>		inference	CPU, GPU	py3

### Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
756306329178.dkr.ecr.sa-1 east-1.amazonaws.com/ xgboost-neo:<tag>		inference		

### Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-1 east-1.amazonaws.com/ object-detection:<tag>		inference, training		

### Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec', region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-1 east-1.amazonaws.com/ object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca', region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-1 east-1.amazonaws.com/ pca:<tag>		inference, training		

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch', region='sa-
east-1', version='1.8.0', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.4@ east-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.4@ east-1.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e0.6@ east-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.6@ east-1.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e0.8@ east-1.amazonaws.com/ sagemaker- pytorch:<tag>		inference	CPU, GPU	py2, py3
520713654638.dkr.e0.8@ east-1.amazonaws.com/ sagemaker- pytorch:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.eif.3a1-east-1.amazonaws.com/pytorch-inference-eia:<tag>		eia	CPU	py3
763104351884.dkr.eif.2a0-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.eif.3a1-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
763104351884.dkr.eif.4a0-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.eif.5a0-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
763104351884.dkr.eif.6a0-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.eif.7a1-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.eif.8a0-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.eif.8a1-east-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
763104351884.dkr.eif.2a0-east-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.eif.3a1-east-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.ecr. <b>40</b> east-1.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.ecr. <b>50</b> east-1.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.ecr. <b>60</b> east-1.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr. <b>70</b> east-1.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr. <b>80</b> east-1.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36
763104351884.dkr.ecr. <b>81</b> east-1.amazonaws.com/ pytorch- training:<tag>		training	CPU, GPU	py3, py36

## Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-1 east-1.amazonaws.com/ randomcutforest:<tag>		inference, training		

## Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn', region='sa-  
east-1', version='0.23-1', image_scope='inference')
```

Registry path	Version	Job types (image scope)		
737474898029.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-scikit-learn:<tag>	0.20.0	inference, training		
737474898029.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-scikit-learn:<tag>	0.23.1	inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-east-1.amazonaws.com/semantic-segmentation:<tag>	1	inference, training		

### Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='sa-east-1')
```

Registry path	Version	Job types (image scope)		
855470959533.dkr.ecr.sa-east-1.amazonaws.com/seq2seq:<tag>	1	inference, training		

### Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark',region='sa-east-1',version='3.0',image_scope='processing')
```

Registry path	Version	Job types (image scope)		
737130764395.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-spark-processing:<tag>	2.4	processing		
737130764395.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-spark-processing:<tag>	3.0	processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving', region='sa-east-1', version='2.4')
```

Registry path	Version	Job types (image scope)		
737474898029.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-sparkml-serving:<tag>	2.2	inference		
737474898029.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-sparkml-serving:<tag>	2.4	inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='sa-east-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-tensorflow-eia:<tag>		eia	CPU	py2
520713654638.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow-scriptmode:<tag>				
520713654638.dkr.e-dr.12.0 east-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2, py3
520713654638.dkr.e-dr.13.1 east-1.amazonaws.com/ sagemaker- tensorflow- scriptmode:<tag>		training	CPU, GPU	py2
520713654638.dkr.e-dr.14.0 east-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.14.0 east-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.14.0 east-1.amazonaws.com/ sagemaker- tensorflow- serving-eia:<tag>		eia	CPU	-
520713654638.dkr.e-dr.14.0 east-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.14.0 east-1.amazonaws.com/ sagemaker- tensorflow- serving:<tag>		inference	CPU, GPU	-
520713654638.dkr.e-dr.10.0 east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e-dr.10.0 east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e1r.4a1-east-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e1r.4a1-east-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.e1r.5a0-east-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e1r.5a0-east-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.e1r.5a0-east-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e1r.5a0-east-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.e1r.7a0-east-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e1r.7a0-east-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.e1r.8a0-east-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
520713654638.dkr.e1r.8a0-east-1.amazonaws.com/sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
520713654638.dkr.e1r.9a0-east-1.amazonaws.com/sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e <u>50</u> .0 east-1.amazonaws.com/ sagemaker- tensorflow:<tag>		training	CPU, GPU	py2
763104351884.dkr.e <u>51</u> .0 east-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>52</u> .0 east-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>53</u> .0 east-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>54</u> .0 east-1.amazonaws.com/ tensorflow- inference- eia:<tag>		eia	CPU	-
763104351884.dkr.e <u>55</u> .0 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>56</u> .0 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>57</u> .0 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>58</u> .2 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>59</u> .3 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e0r.15.4 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0r.15.5 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0r.60 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0r.61 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0r.62 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0r.63 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0r.64 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0r.10 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0r.11 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0r.12 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e0r.13 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0.20</u> -0 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.20</u> -1 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.20</u> -2 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.30</u> -0 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.30</u> -1 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.30</u> -2 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.30</u> -3 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.31</u> -0 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.31</u> -1 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.41</u> -0 east-1.amazonaws.com/ tensorflow- inference:<tag>		inference	CPU, GPU	-
763104351884.dkr.e <u>0.41</u> -1 east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0.41</u> -2 east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.41</u> -3 east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0.41</u> -4 east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e <u>0</u> . <u>15</u> .3 east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3, py37
763104351884.dkr.e <u>0</u> . <u>15</u> .4 east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0</u> . <u>15</u> .5 east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3, py36, py37
763104351884.dkr.e <u>0</u> . <u>00</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0</u> . <u>01</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0</u> . <u>02</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0</u> . <u>03</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0</u> . <u>04</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e <u>0</u> . <u>10</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0</u> . <u>11</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
763104351884.dkr.e <u>0</u> . <u>12</u> east-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
763104351884.dkr.e0.1a3-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
763104351884.dkr.e0.2a0-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.2a1-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.2a2-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.3a0-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.3a1-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.3a2-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
763104351884.dkr.e0.4a1-east-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

### Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow',region='sa-east-1',version='1.0.0',image_scope='training',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.10-east-1.amazonaws.com/sagemaker-rl-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow:coach0.10-<tag>				
520713654638.dkr.e0.10.1east-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11-east-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.0east-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.11.1east-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inference (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow',region='sa-east-1',version='1.15.0',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
756306329178.dkr.e0.15.0east-1.amazonaws.com/sagemaker-neo-tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow',region='sa-east-1',version='0.8.5',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
520713654638.dkr.e0.5a-east-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.5a-east-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6a-east-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6-<tag>		training	CPU, GPU	py3
520713654638.dkr.e0.6a-east-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5-<tag>		training	CPU, GPU	py3

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost',region='sa-east-1',version='1.2-1')
```

Registry path	Version	Job types (image scope)		
737474898029.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-xgboost:<tag>	0.90-1	inference, training		
737474898029.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-xgboost:<tag>	0.90-2	inference, training		
737474898029.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-xgboost:<tag>	1.0-1	inference, training		
737474898029.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-xgboost:<tag>	1.2-1	inference, training		

Registry path	Version	Job types (image scope)		
737474898029.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-xgboost:<tag>	1.2-2	inference, training		
737474898029.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-xgboost:<tag>	1.3-1	inference, training		
855470959533.dkr.ecr.sa-east-1.amazonaws.com/xgboost:<tag>	1	inference, training		

### Docker Registry Paths and Example Code for Amazon GovCloud (US-West) (us-gov-west-1)

The following topics list parameters for each of the algorithms and deep learning containers in this region provided by Amazon SageMaker.

#### Topics

- [BlazingText \(algorithm\) \(p. 1326\)](#)
- [Chainer \(DLC\) \(p. 1326\)](#)
- [Debugger \(algorithm\) \(p. 1327\)](#)
- [DeepAR Forecasting \(algorithm\) \(p. 1327\)](#)
- [Factorization Machines \(algorithm\) \(p. 1327\)](#)
- [Hugging Face \(algorithm\) \(p. 1328\)](#)
- [IP Insights \(algorithm\) \(p. 1328\)](#)
- [Image classification \(algorithm\) \(p. 1329\)](#)
- [Inferentia MXNet \(DLC\) \(p. 1329\)](#)
- [Inferentia PyTorch \(DLC\) \(p. 1329\)](#)
- [K-Means \(algorithm\) \(p. 1330\)](#)
- [KNN \(algorithm\) \(p. 1330\)](#)
- [LDA \(algorithm\) \(p. 1330\)](#)
- [Linear Learner \(algorithm\) \(p. 1331\)](#)
- [MXNet \(DLC\) \(p. 1331\)](#)
- [MXNet Coach \(DLC\) \(p. 1334\)](#)
- [NTM \(algorithm\) \(p. 1334\)](#)
- [Neo Image Classification \(algorithm\) \(p. 1335\)](#)
- [Neo MXNet \(DLC\) \(p. 1335\)](#)
- [Neo PyTorch \(DLC\) \(p. 1335\)](#)
- [Neo Tensorflow \(DLC\) \(p. 1336\)](#)
- [Neo XGBoost \(algorithm\) \(p. 1336\)](#)
- [Object Detection \(algorithm\) \(p. 1337\)](#)
- [Object2Vec \(algorithm\) \(p. 1337\)](#)
- [PCA \(algorithm\) \(p. 1337\)](#)
- [PyTorch \(DLC\) \(p. 1338\)](#)
- [Random Cut Forest \(algorithm\) \(p. 1340\)](#)

- [Scikit-learn \(algorithm\) \(p. 1341\)](#)
- [Semantic Segmentation \(algorithm\) \(p. 1341\)](#)
- [Seq2Seq \(algorithm\) \(p. 1341\)](#)
- [Spark \(algorithm\) \(p. 1342\)](#)
- [SparkML Serving \(algorithm\) \(p. 1342\)](#)
- [Tensorflow \(DLC\) \(p. 1343\)](#)
- [Tensorflow Coach \(DLC\) \(p. 1351\)](#)
- [Tensorflow Inferentia \(DLC\) \(p. 1352\)](#)
- [Tensorflow Ray \(DLC\) \(p. 1352\)](#)
- [XGBoost \(algorithm\) \(p. 1353\)](#)

### [BlazingText \(algorithm\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='blazingtext', region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
226302683700.dkr.ecr.us-1 gov- west-1.amazonaws.com/ blazingtext:<tag>		inference, training		

### [Chainer \(DLC\)](#)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='chainer', region='us-gov-  
west-1', version='5.0.0', py_version='py3', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
246785580436.dkr.ecr.us-1 gov- west-1.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3
246785580436.dkr.ecr.us-1 gov- west-1.amazonaws.com/ sagemaker- chainer:<tag>		inference, training	CPU, GPU	py2, py3
246785580436.dkr.ecr.us-1 gov- west-1.amazonaws.com/		inference, training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
sagemaker-chainer:<tag>				

### Debugger (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='debugger', region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
515509971035.dkr.ecr.us-latest gov-west-1.amazonaws.com/ sagemaker-debugger-rules:<tag>		debugger		

### DeepAR Forecasting (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='forecasting-deepar', region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
226302683700.dkr.ecr.us-1 gov-west-1.amazonaws.com/ forecasting-deepar:<tag>		inference, training		

### Factorization Machines (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='factorization-machines', region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
226302683700.dkr.ecr.us-1 gov-west-1.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
factorization-machines:<tag>				

### Hugging Face (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='huggingface', region='us-gov-west-1', version='4.4.2', image_scope='training', base_framework_version='tensorflow2.4.1')
```

Registry path	Version	Job types (image scope)		
442386744353.dkr.ecr.us-4.4.2.gov-west-1.amazonaws.com/huggingface-pytorch-training:<tag>		training		
442386744353.dkr.ecr.us-4.5.0.gov-west-1.amazonaws.com/huggingface-pytorch-training:<tag>		training		
442386744353.dkr.ecr.us-4.4.2.gov-west-1.amazonaws.com/huggingface-tensorflow-training:<tag>		training		
442386744353.dkr.ecr.us-4.5.0.gov-west-1.amazonaws.com/huggingface-tensorflow-training:<tag>		training		

### IP Insights (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ipinsights', region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
226302683700.dkr.ecr.us-1.gov-		inference, training		

Registry path	Version	Job types (image scope)		
west-1.amazonaws.com/ ipinsights:<tag>				

### Image classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification', region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
226302683700.dkr.ecr.us-1 gov- west-1.amazonaws.com/ image- classification:<tag>		inference, training		

### Inferentia MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-mxnet', region='us-gov-  
west-1', version='1.5.1', instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
263933020539.dkr.ecr.us-1 gov- west-1.amazonaws.com/ sagemaker-neo- mxnet:<tag>		inference	inf	py3

### Inferentia PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-pytorch', region='us-gov-  
west-1', version='1.5.1', py_version='py3')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
263933020539.dkr.ecr.us-1 gov-		inference	inf	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
west-1.amazonaws.com/ sagemaker-neo- pytorch:<tag>				

### K-Means (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='kmeans', region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
226302683700.dkr.ecr.us-1 gov- west-1.amazonaws.com/ kmeans:<tag>		inference, training		

### KNN (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='knn', region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
226302683700.dkr.ecr.us-1 gov- west-1.amazonaws.com/ knn:<tag>		inference, training		

### LDA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='lda', region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
226302683700.dkr.ecr.us-1 gov- west-1.amazonaws.com/ lda:<tag>		inference, training		

## Linear Learner (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='linear-learner', region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
226302683700.dkr.ecr.us-1 gov-west-1.amazonaws.com/ linear-learner:<tag>		inference, training		

## MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='mxnet', region='us-gov-west-1', version='1.4.1', py_version='py3', image_scope='inference',
 instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
246785580436.dkr.ecr.us-1 gov-west-1.amazonaws.com/ sagemaker-mxnet-eia:<tag>		eia	CPU	py2, py3
246785580436.dkr.ecr.us-1 gov-west-1.amazonaws.com/ sagemaker-mxnet-serving-eia:<tag>		eia	CPU	py2, py3
246785580436.dkr.ecr.us-1 gov-west-1.amazonaws.com/ sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2, py3
246785580436.dkr.ecr.us-1 gov-west-1.amazonaws.com/ sagemaker-mxnet-serving:<tag>		inference	CPU, GPU	py2
246785580436.dkr.ecr.us-1 gov-west-1.amazonaws.com/ sagemaker-mxnet:<tag>		inference	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
246785580436.dkr.e0.1 <u>2</u> .1 gov-west-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
246785580436.dkr.e0.0 <u>6</u> - gov-west-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
246785580436.dkr.e0.0 <u>6</u> - gov-west-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
246785580436.dkr.e0.1 <u>0</u> - gov-west-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
246785580436.dkr.e0.1 <u>0</u> - gov-west-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
246785580436.dkr.e0.1 <u>4</u> - gov-west-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
246785580436.dkr.e0.2 <u>4</u> - gov-west-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3
246785580436.dkr.e0.2 <u>4</u> - gov-west-1.amazonaws.com/ sagemaker- mxnet:<tag>		inference	CPU, GPU	py2, py3
246785580436.dkr.e0.3 <u>0</u> - gov-west-1.amazonaws.com/ sagemaker- mxnet:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
246785580436.dkr.edr. <del>4.0</del> gov-west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2, py3
246785580436.dkr.edr. <del>4.1</del> gov-west-1.amazonaws.com/sagemaker-mxnet:<tag>		training	CPU, GPU	py2
442386744353.dkr.edr. <del>4.1</del> gov-west-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
442386744353.dkr.edr. <del>5.1</del> gov-west-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py2, py3
442386744353.dkr.edr. <del>7.0</del> gov-west-1.amazonaws.com/mxnet-inference-eia:<tag>		eia	CPU	py3
442386744353.dkr.edr. <del>4.1</del> gov-west-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
442386744353.dkr.edr. <del>6.0</del> gov-west-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py2, py3
442386744353.dkr.edr. <del>7.0</del> gov-west-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py3
442386744353.dkr.edr. <del>8.0</del> gov-west-1.amazonaws.com/mxnet-inference:<tag>		inference	CPU, GPU	py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
442386744353.dkr.e0r.41\$-gov-west-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
442386744353.dkr.e0r.61\$-gov-west-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py2, py3
442386744353.dkr.e0r.71\$-gov-west-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py3
442386744353.dkr.e0r.81\$-gov-west-1.amazonaws.com/mxnet-training:<tag>		training	CPU, GPU	py37

## MXNet Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-mxnet', region='us-gov-west-1', version='0.11', py_version='py3', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
246785580436.dkr.e0r.11\$-gov-west-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11-<tag>		training	CPU, GPU	py3
246785580436.dkr.e0r.11.0\$-gov-west-1.amazonaws.com/sagemaker-rl-mxnet:coach0.11.0-<tag>		training	CPU, GPU	py3

## NTM (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ntm', region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
226302683700.dkr.ecr.us-1 gov- west-1.amazonaws.com/ ntm:<tag>		inference, training		

### Neo Image Classification (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='image-classification-neo', region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
263933020539.dkr.ecr.us-latest gov- west-1.amazonaws.com/ image-classification- neo:<tag>		inference		

### Neo MXNet (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-mxnet', region='us-gov-  
west-1', version='1.8', py_version='py3', image_scope='inference',  
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
263933020539.dkr.ecr.us- gov- west-1.amazonaws.com/ sagemaker- inference- mxnet:<tag>		inference	CPU, GPU	py3

### Neo PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-pytorch', region='us-gov-west-1', version='1.6', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
263933020539.dkr.ecr.us-gov-west-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
263933020539.dkr.ecr.us-gov-west-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3
263933020539.dkr.ecr.us-gov-west-1.amazonaws.com/sagemaker-inference-pytorch:<tag>		inference	CPU, GPU	py3

## Neo Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='neo-tensorflow', region='us-gov-west-1', version='1.15.3', py_version='py3', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
263933020539.dkr.ecr.us-gov-west-1.amazonaws.com/sagemaker-inference-tensorflow:<tag>		inference	CPU, GPU	py3

## Neo XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost-neo', region='us-gov-west-1')
```

<b>Registry path</b>	<b>Version</b>	<b>Job types (image scope)</b>		
263933020539.dkr.ecr.us-west-1.amazonaws.com/xgboost-neo:<tag>		inference		

## Object Detection (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object-detection', region='us-gov-west-1')
```

<b>Registry path</b>	<b>Version</b>	<b>Job types (image scope)</b>		
226302683700.dkr.ecr.us-west-1.amazonaws.com/object-detection:<tag>		inference, training		

## Object2Vec (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='object2vec', region='us-gov-west-1')
```

<b>Registry path</b>	<b>Version</b>	<b>Job types (image scope)</b>		
226302683700.dkr.ecr.us-west-1.amazonaws.com/object2vec:<tag>		inference, training		

## PCA (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pca', region='us-gov-west-1')
```

<b>Registry path</b>	<b>Version</b>	<b>Job types (image scope)</b>		
226302683700.dkr.ecr.us-west-1.amazonaws.com/pca		inference, training		

Registry path	Version	Job types (image scope)		
west-1.amazonaws.com/pca:<tag>				

## PyTorch (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='pytorch', region='us-gov-west-1', version='1.8.0', py_version='py3', image_scope='inference',
instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
246785580436.dkr.e0.40-gov-west-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
246785580436.dkr.e0.40-gov-west-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
246785580436.dkr.edr.0.0-gov-west-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
246785580436.dkr.edr.0.0-gov-west-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
246785580436.dkr.edr.1.0-gov-west-1.amazonaws.com/sagemaker-pytorch:<tag>		inference	CPU, GPU	py2, py3
246785580436.dkr.edr.1.0-gov-west-1.amazonaws.com/sagemaker-pytorch:<tag>		training	CPU, GPU	py2, py3
442386744353.dkr.edr.3.4-gov-west-1.amazonaws.com/		eia	CPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
pytorch-inference-eia:<tag>				
442386744353.dkr.edr.2 <u>1</u> 0-gov-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
442386744353.dkr.edr.3 <u>1</u> 4-gov-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py2, py3
442386744353.dkr.edr.4 <u>1</u> 0-gov-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
442386744353.dkr.edr.5 <u>1</u> 0-gov-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3
442386744353.dkr.edr.6 <u>1</u> 0-gov-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
442386744353.dkr.edr.7 <u>1</u> 4-gov-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
442386744353.dkr.edr.8 <u>1</u> 0-gov-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
442386744353.dkr.edr.8 <u>1</u> 4-gov-west-1.amazonaws.com/pytorch-inference:<tag>		inference	CPU, GPU	py3, py36
442386744353.dkr.edr.2 <u>1</u> 0-gov-west-1.amazonaws.com/pytorch-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
442386744353.dkr.ecr.us-gov-west-1.amazonaws.com/pytorch-training:<tag>	edr.340	training	CPU, GPU	py2, py3
442386744353.dkr.ecr.440	gov-west-1.amazonaws.com/pytorch-training:<tag>	training	CPU, GPU	py2, py3
442386744353.dkr.ecr.540	gov-west-1.amazonaws.com/pytorch-training:<tag>	training	CPU, GPU	py3
442386744353.dkr.ecr.640	gov-west-1.amazonaws.com/pytorch-training:<tag>	training	CPU, GPU	py3, py36
442386744353.dkr.ecr.740	gov-west-1.amazonaws.com/pytorch-training:<tag>	training	CPU, GPU	py3, py36
442386744353.dkr.ecr.840	gov-west-1.amazonaws.com/pytorch-training:<tag>	training	CPU, GPU	py3, py36
442386744353.dkr.ecr.844	gov-west-1.amazonaws.com/pytorch-training:<tag>	training	CPU, GPU	py3, py36

### Random Cut Forest (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='randomcutforest', region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
226302683700.dkr.ecr.us-1gov-		inference, training		

Registry path	Version	Job types (image scope)		
west-1.amazonaws.com/randomcutforest:<tag>				

### Scikit-learn (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sklearn',region='us-gov-west-1',version='0.23-1',image_scope='inference')
```

Registry path	Version	Job types (image scope)		
414596584902.dkr.ecr.us-0.20.0 gov-west-1.amazonaws.com/ sagemaker-scikit-learn:<tag>		inference, training		
414596584902.dkr.ecr.us-0.23-1 gov-west-1.amazonaws.com/ sagemaker-scikit-learn:<tag>		inference, training		

### Semantic Segmentation (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='semantic-segmentation',region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
226302683700.dkr.ecr.us-1 gov-west-1.amazonaws.com/ semantic-segmentation:<tag>		inference, training		

### Seq2Seq (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='seq2seq',region='us-gov-west-1')
```

Registry path	Version	Job types (image scope)		
226302683700.dkr.ecr.us-1 gov-west-1.amazonaws.com/ seq2seq:<tag>		inference, training		

## Spark (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='spark', region='us-gov-west-1', version='3.0', image_scope='processing')
```

Registry path	Version	Job types (image scope)		
271483468897.dkr.ecr.us-2.4 gov-west-1.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		
271483468897.dkr.ecr.us-3.0 gov-west-1.amazonaws.com/ sagemaker-spark-processing:<tag>		processing		

## SparkML Serving (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='sparkml-serving', region='us-gov-west-1', version='2.4')
```

Registry path	Version	Job types (image scope)		
414596584902.dkr.ecr.us-2.2 gov-west-1.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		
414596584902.dkr.ecr.us-2.4 gov-west-1.amazonaws.com/ sagemaker-sparkml-serving:<tag>		inference		

## Tensorflow (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='tensorflow', region='us-gov-west-1', version='1.12.0', image_scope='inference', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
246785580436.dkr.e-dr.10:0 gov-west-1.amazonaws.com/ sagemaker-tensorflow-eia:<tag>		eia	CPU	py2
246785580436.dkr.e-dr.11:0 gov-west-1.amazonaws.com/ sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2, py3
246785580436.dkr.e-dr.12:0 gov-west-1.amazonaws.com/ sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2, py3
246785580436.dkr.e-dr.13:1 gov-west-1.amazonaws.com/ sagemaker-tensorflow-scriptmode:<tag>		training	CPU, GPU	py2
246785580436.dkr.e-dr.14:0 gov-west-1.amazonaws.com/ sagemaker-tensorflow-serving-eia:<tag>		eia	CPU	-
246785580436.dkr.e-dr.15:0 gov-west-1.amazonaws.com/ sagemaker-tensorflow-serving-eia:<tag>		eia	CPU	-
246785580436.dkr.e-dr.16:0 gov-west-1.amazonaws.com/ sagemaker-		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow-serving-eia:<tag>				
246785580436.dkr.e-dr.1 <u>1</u> :0 gov-west-1.amazonaws.com/ sagemaker-tensorflow- serving:<tag>		inference	CPU, GPU	-
246785580436.dkr.e-dr.1 <u>2</u> :0 gov-west-1.amazonaws.com/ sagemaker-tensorflow- serving:<tag>		inference	CPU, GPU	-
246785580436.dkr.e-dr.1 <u>3</u> :0 gov-west-1.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
246785580436.dkr.e-dr.1 <u>4</u> :0 gov-west-1.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
246785580436.dkr.e-dr.1 <u>5</u> :0 gov-west-1.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
246785580436.dkr.e-dr.1 <u>6</u> :0 gov-west-1.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
246785580436.dkr.e-dr.1 <u>7</u> :0 gov-west-1.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
246785580436.dkr.e-dr.1 <u>8</u> :0 gov-west-1.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2

Registry path	Version	Job types (image scope)	Processor types	Python versions
246785580436.dkr.edr. <del>6.0</del> gov-west-1.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
246785580436.dkr.edr. <del>6.0</del> gov-west-1.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
246785580436.dkr.edr. <del>7.0</del> gov-west-1.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
246785580436.dkr.edr. <del>7.0</del> gov-west-1.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
246785580436.dkr.edr. <del>8.0</del> gov-west-1.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
246785580436.dkr.edr. <del>8.0</del> gov-west-1.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
246785580436.dkr.edr. <del>9.0</del> gov-west-1.amazonaws.com/ sagemaker-tensorflow:<tag>		inference	CPU, GPU	py2
246785580436.dkr.edr. <del>9.0</del> gov-west-1.amazonaws.com/ sagemaker-tensorflow:<tag>		training	CPU, GPU	py2
442386744353.dkr.edr. <del>14.0</del> gov-west-1.amazonaws.com/ tensorflow-inference-eia:<tag>		eia	CPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
442386744353.dkr.e <u>15</u> .0 gov-west-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
442386744353.dkr.e <u>0.01</u> .0 gov-west-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
442386744353.dkr.e <u>0.31</u> .0 gov-west-1.amazonaws.com/tensorflow-inference-eia:<tag>		eia	CPU	-
442386744353.dkr.e <u>0.13</u> .0 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e <u>0.14</u> .0 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e <u>0.15</u> .0 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e <u>0.15</u> .2 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e <u>0.15</u> .3 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
442386744353.dkr.e <u>0</u> .1 <u>5</u> .4 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e <u>0</u> .1 <u>5</u> .5 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e <u>0</u> .0 <u>6</u> gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e <u>0</u> .0 <u>5</u> gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e <u>0</u> .0 <u>4</u> gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e <u>0</u> .0 <u>3</u> gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e <u>0</u> .0 <u>2</u> gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e <u>0</u> .0 <u>1</u> gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e <u>0</u> .0 <u>0</u> gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
442386744353.dkr.e0r.1u2 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e0r.1u3 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e0r.2u0 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e0r.2u1 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e0r.2u2 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e0r.3u0 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e0r.3u1 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e0r.3u2 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-
442386744353.dkr.e0r.4u1 gov-west-1.amazonaws.com/tensorflow-inference:<tag>		inference	CPU, GPU	-

Registry path	Version	Job types (image scope)	Processor types	Python versions
442386744353.dkr.e <u>dr.13</u> :1 gov-west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3
442386744353.dkr.e <u>dr.14</u> :0 gov-west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
442386744353.dkr.e <u>dr.15</u> :0 gov-west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
442386744353.dkr.e <u>dr.15</u> :2 gov-west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37
442386744353.dkr.e <u>dr.15</u> :3 gov-west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3, py37
442386744353.dkr.e <u>dr.15</u> :4 gov-west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
442386744353.dkr.e <u>dr.15</u> :5 gov-west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py3, py36, py37
442386744353.dkr.e <u>0r.01</u> :0 gov-west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3
442386744353.dkr.e <u>0r.01</u> :1 gov-west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py2, py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
442386744353.dkr.e <u>0.02</u> gov-west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
442386744353.dkr.e <u>0.03</u> gov-west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
442386744353.dkr.e <u>0.04</u> gov-west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
442386744353.dkr.e <u>0.10</u> gov-west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
442386744353.dkr.e <u>0.14</u> gov-west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py2, py3
442386744353.dkr.e <u>0.12</u> gov-west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
442386744353.dkr.e <u>0.15</u> gov-west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py3
442386744353.dkr.e <u>0.20</u> gov-west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py37
442386744353.dkr.e <u>0.24</u> gov-west-1.amazonaws.com/ tensorflow- training:<tag>		training	CPU, GPU	py37

Registry path	Version	Job types (image scope)	Processor types	Python versions
442386744353.dkr.e0.212-gov-west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
442386744353.dkr.e0.310-gov-west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
442386744353.dkr.e0.314-gov-west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
442386744353.dkr.e0.312-gov-west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37
442386744353.dkr.e0.414-gov-west-1.amazonaws.com/tensorflow-training:<tag>		training	CPU, GPU	py37

## Tensorflow Coach (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='coach-tensorflow', region='us-gov-west-1', version='1.0.0', image_scope='training', instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
246785580436.dkr.e0.10-gov-west-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.10-<tag>		training	CPU, GPU	py3
246785580436.dkr.e0.10-1-gov-west-1.amazonaws.com/sagemaker-rl-		training	CPU, GPU	py3

Registry path	Version	Job types (image scope)	Processor types	Python versions
tensorflow:coach0.10.1-<tag>				
246785580436.dkr.e0.1\$-gov-west-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11-<tag>		training	CPU, GPU	py3
246785580436.dkr.e0.1\$-gov-west-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.0-<tag>		training	CPU, GPU	py3
246785580436.dkr.e0.1\$-gov-west-1.amazonaws.com/sagemaker-rl-tensorflow:coach0.11.1-<tag>		training	CPU, GPU	py3

## Tensorflow Inferentia (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='inferentia-tensorflow',region='us-gov-west-1',version='1.15.0',instance_type='ml.inf1.6xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
263933020539.dkr.e0.1\$-gov-west-1.amazonaws.com/sagemaker-neo-tensorflow:<tag>		inference	inf	py3

## Tensorflow Ray (DLC)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='ray-tensorflow',region='us-gov-west-1',version='0.8.5',instance_type='ml.c5.4xlarge')
```

Registry path	Version	Job types (image scope)	Processor types	Python versions
246785580436.dkr.e0.5s-gov-west-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5-<tag>		training	CPU, GPU	py3
246785580436.dkr.e0.5s-gov-west-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.5.3-<tag>		training	CPU, GPU	py3
246785580436.dkr.e0.6s-gov-west-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6-<tag>		training	CPU, GPU	py3
246785580436.dkr.e0.6s-gov-west-1.amazonaws.com/sagemaker-rl-tensorflow:ray0.6.5-<tag>		training	CPU, GPU	py3

## XGBoost (algorithm)

SageMaker Python SDK example to retrieve registry path.

```
from sagemaker import image_uris
image_uris.retrieve(framework='xgboost', region='us-gov-west-1', version='1.2-1')
```

Registry path	Version	Job types (image scope)		
226302683700.dkr.ecr.us-1gov-west-1.amazonaws.com/xgboost:<tag>		inference, training		
414596584902.dkr.ecr.us-0.90-1gov-west-1.amazonaws.com/sagemaker-xgboost:<tag>		inference, training		
414596584902.dkr.ecr.us-0.90-2gov-west-1.amazonaws.com/		inference, training		

Registry path	Version	Job types (image scope)		
sagemaker-xgboost:<tag>				
414596584902.dkr.ecr.us-west-1.amazonaws.com/sagemaker-xgboost:<tag>	1.0-1	inference, training		
414596584902.dkr.ecr.us-west-1.amazonaws.com/sagemaker-xgboost:<tag>	1.2-1	inference, training		
414596584902.dkr.ecr.us-west-1.amazonaws.com/sagemaker-xgboost:<tag>	1.2-2	inference, training		
414596584902.dkr.ecr.us-west-1.amazonaws.com/sagemaker-xgboost:<tag>	1.3-1	inference, training		

## Common Data Formats for Built-in Algorithms

The following topics explain the data formats for the algorithms provided by Amazon SageMaker.

### Topics

- [Common Data Formats for Training \(p. 1354\)](#)
- [Common Data Formats for Inference \(p. 1358\)](#)

### Common Data Formats for Training

To prepare for training, you can preprocess your data using a variety of Amazon services, including Amazon Glue, Amazon EMR, Amazon Redshift, Amazon Relational Database Service, and Amazon Athena. After preprocessing, publish the data to an Amazon S3 bucket. For training, the data need to go through a series of conversions and transformations, including:

- Training data serialization (handled by you)
- Training data deserialization (handled by the algorithm)
- Training model serialization (handled by the algorithm)
- Trained model deserialization (optional, handled by you)

When using Amazon SageMaker in the training portion of the algorithm, make sure to upload all data at once. If more data is added to that location, a new training call would need to be made to construct a brand new model.

### Topics

- [Content Types Supported by Built-In Algorithms \(p. 1355\)](#)
- [Using Pipe Mode \(p. 1355\)](#)
- [Using CSV Format \(p. 1355\)](#)
- [Using RecordIO Format \(p. 1356\)](#)
- [Trained Model Deserialization \(p. 1358\)](#)

## Content Types Supported by Built-In Algorithms

The following table lists some of the commonly supported `ContentType` values and the algorithms that use them:

### ContentTypes for Built-in Algorithms

ContentType	Algorithm
application/x-image	Object Detection Algorithm, Semantic Segmentation
application/x-recordio	Object Detection Algorithm
application/x-recordio-protobuf	Factorization Machines, K-Means, k-NN, Latent Dirichlet Allocation, Linear Learner, NTM, PCA, RCF, Sequence-to-Sequence
application/jsonlines	BlazingText, DeepAR
image/jpeg	Object Detection Algorithm, Semantic Segmentation
image/png	Object Detection Algorithm, Semantic Segmentation
text/csv	IP Insights, K-Means, k-NN, Latent Dirichlet Allocation, Linear Learner, NTM, PCA, RCF, XGBoost
text/libsvm	XGBoost

For a summary of the parameters used by each algorithm, see the documentation for the individual algorithms or this [table](#).

### Using Pipe Mode

In *Pipe mode*, your training job streams data directly from Amazon Simple Storage Service (Amazon S3). Streaming can provide faster start times for training jobs and better throughput. This is in contrast to *File mode*, in which your data from Amazon S3 is stored on the training instance volumes. File mode uses disk space to store both your final model artifacts and your full training dataset. By streaming in your data directly from Amazon S3 in Pipe mode, you reduce the size of Amazon Elastic Block Store volumes of your training instances. Pipe mode needs only enough disk space to store your final model artifacts. See the [AlgorithmSpecification](#) for additional details on the training input mode.

### Using CSV Format

Many Amazon SageMaker algorithms support training with data in CSV format. To use data in CSV format for training, in the input data channel specification, specify `text/csv` as the `ContentType`. Amazon SageMaker requires that a CSV file does not have a header record and that the target variable is in the first column. To run unsupervised learning algorithms that don't have a target, specify the number of label columns in the content type. For example, in this case '`content_type=text/csv;label_size=0`'. For a notebook example that uses CSV format, see [Breast Cancer Prediction](#). For more information, see [Now use Pipe mode with CSV datasets for faster training on Amazon SageMaker built-in algorithms](#).

## Using RecordIO Format

In the protobuf recordIO format, SageMaker converts each observation in the dataset into a binary representation as a set of 4-byte floats, then loads it in the protobuf values field. If you are using Python for your data preparation, we strongly recommend that you use these existing transformations. However, if you are using another language, the protobuf definition file below provides the schema that you use to convert your data into SageMaker protobuf format.

**Note**

For an example that shows how to convert the commonly used numpy array into the protobuf recordIO format, see [An Introduction to Factorization Machines with MNIST](#).

```
syntax = "proto2";

package aialgs.data;

option java_package = "com.amazonaws.aialgorithms.proto";
option java_outer_classname = "RecordProtos";

// A sparse or dense rank-R tensor that stores data as doubles (float64).
message Float32Tensor {
    // Each value in the vector. If keys is empty, this is treated as a
    // dense vector.
    repeated float values = 1 [packed = true];

    // If key is not empty, the vector is treated as sparse, with
    // each key specifying the location of the value in the sparse vector.
    repeated uint64 keys = 2 [packed = true];

    // An optional shape that allows the vector to represent a matrix.
    // For example, if shape = [ 10, 20 ], floor(keys[i] / 20) gives the row,
    // and keys[i] % 20 gives the column.
    // This also supports n-dimensional tensors.
    // Note: If the tensor is sparse, you must specify this value.
    repeated uint64 shape = 3 [packed = true];
}

// A sparse or dense rank-R tensor that stores data as doubles (float64).
message Float64Tensor {
    // Each value in the vector. If keys is empty, this is treated as a
    // dense vector.
    repeated double values = 1 [packed = true];

    // If this is not empty, the vector is treated as sparse, with
    // each key specifying the location of the value in the sparse vector.
    repeated uint64 keys = 2 [packed = true];

    // An optional shape that allows the vector to represent a matrix.
    // For example, if shape = [ 10, 20 ], floor(keys[i] / 10) gives the row,
    // and keys[i] % 20 gives the column.
    // This also supports n-dimensional tensors.
    // Note: If the tensor is sparse, you must specify this value.
    repeated uint64 shape = 3 [packed = true];
}

// A sparse or dense rank-R tensor that stores data as 32-bit ints (int32).
message Int32Tensor {
    // Each value in the vector. If keys is empty, this is treated as a
    // dense vector.
    repeated int32 values = 1 [packed = true];

    // If this is not empty, the vector is treated as sparse with
    // each key specifying the location of the value in the sparse vector.
    repeated uint64 keys = 2 [packed = true];
```

```

// An optional shape that allows the vector to represent a matrix.
// For example, if shape = [ 10, 20 ], floor(keys[i] / 10) gives the row,
// and keys[i] % 20 gives the column.
// This also supports n-dimensional tensors.
// Note: If the tensor is sparse, you must specify this value.
repeated uint64 shape = 3 [packed = true];
}

// Support for storing binary data for parsing in other ways (such as JPEG/etc).
// This is an example of another type of value and may not immediately be supported.
message Bytes {
    repeated bytes value = 1;

    // If the content type of the data is known, stores it.
    // This allows for the possibility of using decoders for common formats
    // in the future.
    optional string content_type = 2;
}

message Value {
    oneof value {
        // The numbering assumes the possible use of:
        // - float16, float128
        // - int8, int16, int32
        Float32Tensor float32_tensor = 2;
        Float64Tensor float64_tensor = 3;
        Int32Tensor int32_tensor = 7;
        Bytes bytes = 9;
    }
}

message Record {
    // Map from the name of the feature to the value.
    //
    // For vectors and libsvm-like datasets,
    // a single feature with the name `values`
    // should be specified.
    map<string, Value> features = 1;

    // An optional set of labels for this record.
    // Similar to the features field above, the key used for
    // generic scalar / vector labels should be 'values'.
    map<string, Value> label = 2;

    // A unique identifier for this record in the dataset.
    //
    // Whilst not necessary, this allows better
    // debugging where there are data issues.
    //
    // This is not used by the algorithm directly.
    optional string uid = 3;

    // Textual metadata describing the record.
    //
    // This may include JSON-serialized information
    // about the source of the record.
    //
    // This is not used by the algorithm directly.
    optional string metadata = 4;

    // An optional serialized JSON object that allows per-record
    // hyper-parameters/configuration/other information to be set.
    //
    // The meaning/interpretation of this field is defined by
    // the algorithm author and may not be supported.
    //
}

```

```
// This is used to pass additional inference configuration
// when batch inference is used (e.g. types of scores to return).
optional string configuration = 5;
}
```

After creating the protocol buffer, store it in an Amazon S3 location that Amazon SageMaker can access and that can be passed as part of `InputDataConfig` in `create_training_job`.

**Note**

For all Amazon SageMaker algorithms, the `ChannelName` in `InputDataConfig` must be set to `train`. Some algorithms also support a validation or test input channels. These are typically used to evaluate the model's performance by using a hold-out dataset. Hold-out datasets are not used in the initial training but can be used to further tune the model.

## Trained Model Deserialization

Amazon SageMaker models are stored as `model.tar.gz` in the S3 bucket specified in `OutputDataConfig.S3OutputPath` parameter of the `create_training_job` call. You can specify most of these model artifacts when creating a hosting model. You can also open and review them in your notebook instance. When `model.tar.gz` is untarred, it contains `model_algo-1`, which is a serialized Apache MXNet object. For example, you use the following to load the k-means model into memory and view it:

```
import mxnet as mx
print(mx.ndarray.load('model_algo-1'))
```

## Common Data Formats for Inference

Amazon SageMaker algorithms accept and produce several different MIME types for the HTTP payloads used in retrieving online and mini-batch predictions. You can use various Amazon services to transform or preprocess records prior to running inference. At a minimum, you need to convert the data for the following:

- Inference request serialization (handled by you)
- Inference request deserialization (handled by the algorithm)
- Inference response serialization (handled by the algorithm)
- Inference response deserialization (handled by you)

### Topics

- [Convert Data for Inference Request Serialization \(p. 1358\)](#)
- [Convert Data for Inference Response Deserialization \(p. 1359\)](#)
- [Common Request Formats for All Algorithms \(p. 1361\)](#)
- [Use Batch Transform with Built-in Algorithms \(p. 1362\)](#)

## Convert Data for Inference Request Serialization

Content type options for Amazon SageMaker algorithm inference requests include: `text/csv`, `application/json`, and `application/x-recordio-protobuf`. Algorithms that don't support all of these types can support other types. XGBoost, for example, only supports `text/csv` from this list, but also supports `text/libsvm`.

For `text/csv`, the value for the `Body` argument to `invoke_endpoint` should be a string with commas separating the values for each feature. For example, a record for a model with four features might look like `1.5,16.0,14,23.0`. Any transformations performed on the training data should also be performed on the data before obtaining inference. The order of the features matters and must remain unchanged.

`application/json` is significantly more flexible and provides multiple possible formats for developers to use in their applications. At a high level, in JavaScript, the payload might look like the following:

```
let request = {
    // Instances might contain multiple rows that predictions are sought for.
    "instances": [
        {
            // Request and algorithm specific inference parameters.
            "configuration": {},
            // Data in the specific format required by the algorithm.
            "data": {
                "<field name>": dataElement
            }
        }
    ]
}
```

You have the following options for specifying the `dataElement`:

#### Protocol buffers equivalent

```
// Has the same format as the protocol buffers implementation described for training.
let dataElement = {
    "keys": [],
    "values": [],
    "shape": []
}
```

#### Simple numeric vector

```
// An array containing numeric values is treated as an instance containing a
// single dense vector.
let dataElement = [1.5, 16.0, 14.0, 23.0]

// It will be converted to the following representation by the SDK.
let converted = {
    "features": {
        "values": dataElement
    }
}
```

#### For multiple records

```
let request = {
    "instances": [
        // First instance.
        {
            "features": [ 1.5, 16.0, 14.0, 23.0 ]
        },
        // Second instance.
        {
            "features": [ -2.0, 100.2, 15.2, 9.2 ]
        }
    ]
}
```

#### Convert Data for Inference Response Deserialization

Amazon SageMaker algorithms return JSON in several layouts. At a high level, the structure is:

```
let response = {
```

```

    "predictions": [
        // Fields in the response object are defined on a per algorithm-basis.
    ]
}

```

The fields that are included in predictions differ across algorithms. The following are examples of output for the k-means algorithm.

### Single-record inference

```

let response = {
    "predictions": [
        {
            "closest_cluster": 5,
            "distance_to_cluster": 36.5
        }
    ]
}

```

### Multi-record inference

```

let response = {
    "predictions": [
        // First instance prediction.
        {
            "closest_cluster": 5,
            "distance_to_cluster": 36.5
        },
        // Second instance prediction.
        {
            "closest_cluster": 2,
            "distance_to_cluster": 90.3
        }
    ]
}

```

### Multi-record inference with protobuf input

```

{
    "features": [],
    "label": {
        "closest_cluster": {
            "values": [ 5.0 ] // e.g. the closest centroid/cluster was 1.0
        },
        "distance_to_cluster": {
            "values": [ 36.5 ]
        }
    },
    "uid": "abc123",
    "metadata": "{ \"created_at\": '2017-06-03' }"
}

```

SageMaker algorithms also support the JSONLINES format, where the per-record response content is same as that in JSON format. The multi-record structure is a concatenation of per-record response objects separated by newline characters. The response content for the built-in KMeans algorithm for 2 input data points is:

```

{"distance_to_cluster": 23.40593910217285, "closest_cluster": 0.0}
{"distance_to_cluster": 27.250282287597656, "closest_cluster": 0.0}

```

While running batch transform, we recommended using the jsonlines response type by setting the Accept field in the CreateTransformJobRequest to application/jsonlines.

## Common Request Formats for All Algorithms

Most algorithms use several of the following inference request formats.

### JSON Request Format

**Content type:** application/JSON

#### Dense format

```
let request = {
    "instances": [
        {
            "features": [1.5, 16.0, 14.0, 23.0]
        }
    ]
}

let request = {
    "instances": [
        {
            "data": {
                "features": {
                    "values": [ 1.5, 16.0, 14.0, 23.0]
                }
            }
        }
    ]
}
```

#### Sparse format

```
{
    "instances": [
        {"data": {"features": {
            "keys": [26, 182, 232, 243, 431],
            "shape": [2000],
            "values": [1, 1, 1, 4, 1]
        }}
        },
        {"data": {"features": {
            "keys": [0, 182, 232, 243, 431],
            "shape": [2000],
            "values": [13, 1, 1, 4, 1]
        }}
        },
        ...
    ]
}
```

### JSONLINES Request Format

**Content type:** application/JSONLINES

#### Dense format

A single record in dense format can be represented as either:

```
{ "features": [1.5, 16.0, 14.0, 23.0] }
```

or:

```
{ "data": { "features": { "values": [ 1.5, 16.0, 14.0, 23.0] } } }
```

### Sparse Format

A single record in sparse format is represented as:

```
{"data": {"features": { "keys": [26, 182, 232, 243, 431], "shape": [2000], "values": [1, 1, 1, 4, 1] } } }
```

Multiple records are represented as a concatenation of the above single-record representations, separated by newline characters:

```
{"data": {"features": { "keys": [0, 1, 3], "shape": [4], "values": [1, 4, 1] } } }
{"data": {"features": { "values": [ 1.5, 16.0, 14.0, 23.0] } }
{ "features": [1.5, 16.0, 14.0, 23.0] }
```

### CSV Request Format

**Content type:** text/CSV; label\_size=0

#### Note

CSV support is not available for factorization machines.

### RECORDIO Request Format

Content type: application/x-recordio-protobuf

### Use Batch Transform with Built-in Algorithms

While running batch transform, we recommended using the JSONLINES response type instead of JSON, if supported by the algorithm. This is accomplished by setting the `Accept` field in the `CreateTransformJobRequest` to `application/jsonlines`.

When you create a transform job, the `SplitType` must be set according to the `ContentType` of the input data. Similarly, depending on the `Accept` field in the `CreateTransformJobRequest`, `AssembleWith` must be set accordingly. Please use the following table to help appropriately set these fields:

ContentType	Recommended SplitType
application/x-recordio-protobuf	RecordIO
text/csv	Line
application/jsonlines	Line
application/json	None
application/x-image	None
image/*	None
Accept	Recommended AssembleWith
application/x-recordio-protobuf	None
application/json	None
application/jsonlines	Line

For more information on response formats for specific algorithms, see the following:

- [DeepAR Inference Formats \(p. 1386\)](#)
- [Factorization Machines Response Formats \(p. 1397\)](#)
- [IP Insights Inference Data Formats \(p. 1418\)](#)
- [K-Means Response Formats \(p. 1427\)](#)
- [k-NN Request and Response Formats \(p. 1434\)](#)
- [Linear learner response formats \(p. 1456\)](#)
- [NTM Response Formats \(p. 1462\)](#)
- [Data Formats for Object2Vec Inference \(p. 1477\)](#)
- [Encoder Embeddings for Object2Vec \(p. 1478\)](#)
- [PCA Response Formats \(p. 1493\)](#)
- [RCF Response Formats \(p. 1500\)](#)

## Instance Types for Built-in Algorithms

For training and hosting Amazon SageMaker algorithms, we recommend using the following Amazon EC2 instance types:

- ml.m5.xlarge, ml.m5.4xlarge, and ml.m5.10xlarge
- ml.c5.xlarge, ml.c5.2xlarge, and ml.c5.8xlarge
- ml.p3.xlarge, ml.p3.8xlarge, and ml.p3.16xlarge

Most Amazon SageMaker algorithms have been engineered to take advantage of GPU computing for training. Despite higher per-instance costs, GPUs train more quickly, making them more cost effective. Exceptions are noted in this guide.

The size and type of data can have a great effect on which hardware configuration is most effective. When the same model is trained on a recurring basis, initial testing across a spectrum of instance types can discover configurations that are more cost-effective in the long run. Additionally, algorithms that train most efficiently on GPUs might not require GPUs for efficient inference. Experiment to determine the most cost effectiveness solution.

For more information on SageMaker hardware specifications, see [Amazon SageMaker ML Instance Types](#).

## Logs for Built-in Algorithms

Amazon SageMaker algorithms produce Amazon CloudWatch logs, which provide detailed information on the training process. To see the logs, in the Amazon management console, choose **CloudWatch**, choose **Logs**, and then choose the `/aws/sagemaker/TrainingJobs` **log group**. Each training job has one log stream per node on which it was trained. The log stream's name begins with the value specified in the `TrainingJobName` parameter when the job was created.

### Note

If a job fails and logs do not appear in CloudWatch, it's likely that an error occurred before the start of training. Reasons include specifying the wrong training image or S3 location.

The contents of logs vary by algorithms. However, you can typically find the following information:

- Confirmation of arguments provided at the beginning of the log
- Errors that occurred during training
- Measurement of an algorithm's accuracy or numerical performance
- Timings for the algorithm and any major stages within the algorithm

## Common Errors

If a training job fails, some details about the failure are provided by the `FailureReason` return value in the training job description, as follows:

```
sage = boto3.client('sagemaker')
sage.describe_training_job(TrainingJobName=job_name)[ 'FailureReason' ]
```

Others are reported only in the CloudWatch logs. Common errors include the following:

1. Misspecifying a hyperparameter or specifying a hyperparameter that is invalid for the algorithm.

### From the CloudWatch Log

```
[10/16/2017 23:45:17 ERROR 139623806805824 train.py:48]
Additional properties are not allowed (u'mini_batch_size' was
unexpected)
```

2. Specifying an invalid value for a hyperparameter.

### FailureReason

```
AlgorithmError: u'abc' is not valid under any of the given
schemas\n\nFailed validating u'oneOf' in
schema[u'properties'][u'feature_dim']:\n    {u'oneOf':
[ {u'pattern': u'^([1-9][0-9]*)$', u'type': u'string'},\n     {u'minimum': 1, u'type': u'integer'}]}\\
```

### FailureReason

```
[10/16/2017 23:57:17 ERROR 140373086025536 train.py:48] u'abc'
is not valid under any of the given schemas
```

3. Inaccurate protobuf file format.

### From the CloudWatch log

```
[10/17/2017 18:01:04 ERROR 140234860816192 train.py:48] cannot
copy sequence with size 785 to array axis with dimension 784
```

## BlazingText algorithm

The Amazon SageMaker BlazingText algorithm provides highly optimized implementations of the Word2vec and text classification algorithms. The Word2vec algorithm is useful for many downstream natural language processing (NLP) tasks, such as sentiment analysis, named entity recognition, machine translation, etc. Text classification is an important task for applications that perform web searches, information retrieval, ranking, and document classification.

The Word2vec algorithm maps words to high-quality distributed vectors. The resulting vector representation of a word is called a *word embedding*. Words that are semantically similar correspond to vectors that are close together. That way, word embeddings capture the semantic relationships between words.

Many natural language processing (NLP) applications learn word embeddings by training on large collections of documents. These pretrained vector representations provide information about semantics and word distributions that typically improves the generalizability of other models that are later trained

on a more limited amount of data. Most implementations of the Word2vec algorithm are not optimized for multi-core CPU architectures. This makes it difficult to scale to large datasets.

With the BlazingText algorithm, you can scale to large datasets easily. Similar to Word2vec, it provides the Skip-gram and continuous bag-of-words (CBOW) training architectures. BlazingText's implementation of the supervised multi-class, multi-label text classification algorithm extends the fastText text classifier to use GPU acceleration with custom [CUDA](#) kernels. You can train a model on more than a billion words in a couple of minutes using a multi-core CPU or a GPU. And, you achieve performance on par with the state-of-the-art deep learning text classification algorithms.

The BlazingText algorithm is not parallelizable. For more information on parameters related to training, see [Docker Registry Paths for SageMaker Built-in Algorithms](#).

The SageMaker BlazingText algorithms provides the following features:

- Accelerated training of the fastText text classifier on multi-core CPUs or a GPU and Word2Vec on GPUs using highly optimized CUDA kernels. For more information, see [BlazingText: Scaling and Accelerating Word2Vec using Multiple GPUs](#).
- **Enriched Word Vectors with Subword Information** by learning vector representations for character n-grams. This approach enables BlazingText to generate meaningful vectors for out-of-vocabulary (OOV) words by representing their vectors as the sum of the character n-gram (subword) vectors.
- A `batch_skipgram` mode for the Word2Vec algorithm that allows faster training and distributed computation across multiple CPU nodes. The `batch_skipgram` mode does mini-batching using the Negative Sample Sharing strategy to convert level-1 BLAS operations into level-3 BLAS operations. This efficiently leverages the multiply-add instructions of modern architectures. For more information, see [Parallelizing Word2Vec in Shared and Distributed Memory](#).

To summarize, the following modes are supported by BlazingText on different types instances:

Modes	Word2Vec (Unsupervised Learning)	Text Classification (Supervised Learning)
Single CPU instance	cbow  Skip-gram  Batch Skip-gram	supervised
Single GPU instance (with 1 or more GPUs)	cbow  Skip-gram	supervised with one GPU
Multiple CPU instances	Batch Skip-gram	None

For more information about the mathematics behind BlazingText, see [BlazingText: Scaling and Accelerating Word2Vec using Multiple GPUs](#).

### Topics

- [Input/Output Interface for the BlazingText Algorithm \(p. 1366\)](#)
- [EC2 Instance Recommendation for the BlazingText Algorithm \(p. 1368\)](#)
- [BlazingText Sample Notebooks \(p. 1369\)](#)
- [BlazingText Hyperparameters \(p. 1369\)](#)
- [Tune a BlazingText Model \(p. 1373\)](#)

## Input/Output Interface for the BlazingText Algorithm

The BlazingText algorithm expects a single preprocessed text file with space-separated tokens. Each line in the file should contain a single sentence. If you need to train on multiple text files, concatenate them into one file and upload the file in the respective channel.

### Training and Validation Data Format

#### Training and Validation Data Format for the Word2Vec Algorithm

For Word2Vec training, upload the file under the *train* channel. No other channels are supported. The file should contain a training sentence per line.

#### Training and Validation Data Format for the Text Classification Algorithm

For supervised mode, you can train with file mode or with the augmented manifest text format.

##### Train with File Mode

For supervised mode, the training/validation file should contain a training sentence per line along with the labels. Labels are words that are prefixed by the string `__label__`. Here is an example of a training/validation file:

```
__label__4 linux ready for prime time , intel says , despite all the linux hype , the
open-source movement has yet to make a huge splash in the desktop market . that may be
about to change , thanks to chipmaking giant intel corp .

__label__2 bowled by the slower one again , kolkata , november 14 the past caught up with
sourav ganguly as the indian skippers return to international cricket was short lived .
```

##### Note

The order of labels within the sentence doesn't matter.

Upload the training file under the *train* channel, and optionally upload the validation file under the *validation* channel.

##### Train with Augmented Manifest Text Format

The supervised mode also supports the augmented manifest format, which enables you to do training in pipe mode without needing to create RecordIO files. While using the format, an S3 manifest file needs to be generated that contains the list of sentences and their corresponding labels. The manifest file format should be in [JSON Lines](#) format in which each line represents one sample. The sentences are specified using the `source` tag and the label can be specified using the `label` tag. Both `source` and `label` tags should be provided under the `AttributeNames` parameter value as specified in the request.

```
{"source":"linux ready for prime time , intel says , despite all the linux hype",
 "label":1}
{"source":"bowled by the slower one again , kolkata , november 14 the past caught up with
sourav ganguly", "label":2}
```

Multi-label training is also supported by specifying a JSON array of labels.

```
{"source":"linux ready for prime time , intel says , despite all the linux hype", "label":
 [1, 3]}
 {"source":"bowled by the slower one again , kolkata , november 14 the past caught up with
sourav ganguly", "label": [2, 4, 5]}
```

For more information on augmented manifest files, see [Provide Dataset Metadata to Training Jobs with an Augmented Manifest File \(p. 1864\)](#).

## Model Artifacts and Inference

### Model Artifacts for the Word2Vec Algorithm

For Word2Vec training, the model artifacts consist of `vectors.txt`, which contains words-to-vectors mapping, and `vectors.bin`, a binary used by BlazingText for hosting, inference, or both. `vectors.txt` stores the vectors in a format that is compatible with other tools like Gensim and Spacy. For example, a Gensim user can run the following commands to load the `vectors.txt` file:

```
from gensim.models import KeyedVectors
word_vectors = KeyedVectors.load_word2vec_format('vectors.txt', binary=False)
word_vectors.most_similar(positive=['woman', 'king'], negative=['man'])
word_vectors.doesnt_match("breakfast cereal dinner lunch".split())
```

If the evaluation parameter is set to `True`, an additional file, `eval.json`, is created. This file contains the similarity evaluation results (using Spearman's rank correlation coefficients) on WS-353 dataset. The number of words from the WS-353 dataset that aren't there in the training corpus are reported.

For inference requests, the model accepts a JSON file containing a list of strings and returns a list of vectors. If the word is not found in vocabulary, inference returns a vector of zeros. If subwords is set to `True` during training, the model is able to generate vectors for out-of-vocabulary (OOV) words.

### Sample JSON Request

Mime-type: `application/json`

```
{
  "instances": ["word1", "word2", "word3"]
}
```

### Model Artifacts for the Text Classification Algorithm

Training with supervised outputs creates a `model.bin` file that can be consumed by BlazingText hosting. For inference, the BlazingText model accepts a JSON file containing a list of sentences and returns a list of corresponding predicted labels and probability scores. Each sentence is expected to be a string with space-separated tokens, words, or both.

### Sample JSON Request

Mime-type: `application/json`

```
{
  "instances": ["the movie was excellent", "i did not like the plot ."]
}
```

By default, the server returns only one prediction, the one with the highest probability. For retrieving the top  $k$  predictions, you can set  $k$  in the configuration, as follows:

```
{
  "instances": ["the movie was excellent", "i did not like the plot ."],
  "configuration": {"k": 2}
}
```

For BlazingText, the `content-type` and `accept` parameters must be equal. For batch transform, they both need to be `application/jsonlines`. If they differ, the `Accept` field is ignored. The format for input follows:

```
content-type: application/jsonlines

{"source": "source_0"}
 {"source": "source_1"}

if you need to pass the value of k for top-k, then you can do it in the following way:

 {"source": "source_0", "k": 2}
 {"source": "source_1", "k": 3}
```

The format for output follows:

```
accept: application/jsonlines

 {"prob": [prob_1], "label": ["__label__1"]}
 {"prob": [prob_1], "label": ["__label__1"]}

If you have passed the value of k to be more than 1, then response will be in this format:

 {"prob": [prob_1, prob_2], "label": ["__label__1", "__label__2"]}
 {"prob": [prob_1, prob_2], "label": ["__label__1", "__label__2"]}
```

For both supervised (text classification) and unsupervised (Word2Vec) modes, the binaries (\*.bin) produced by BlazingText can be cross-consumed by fastText and vice versa. You can use binaries produced by BlazingText by fastText. Likewise, you can host the model binaries created with fastText using BlazingText.

Here is an example of how to use a model generated with BlazingText with fastText:

```
#Download the model artifact from S3
aws s3 cp s3://<YOUR_S3_BUCKET>/<PREFIX>/model.tar.gz model.tar.gz

#Unzip the model archive
tar -xzf model.tar.gz

#Use the model archive with fastText
fasttext predict ./model.bin test.txt
```

However, the binaries are only supported when training on CPU and single GPU; training on multi-GPU will not produce binaries.

For more details on dataset formats and model hosting, see the example notebooks [Text Classification with the BlazingText Algorithm](#), [FastText Models](#), and [Generating Subword Embeddings with the Word2Vec Algorithm](#).

## EC2 Instance Recommendation for the BlazingText Algorithm

For cbow and skipgram modes, BlazingText supports single CPU and single GPU instances. Both of these modes support learning of subwords embeddings. To achieve the highest speed without compromising accuracy, we recommend that you use an ml.p3.2xlarge instance.

For batch\_skipgram mode, BlazingText supports single or multiple CPU instances. When training on multiple instances, set the value of the `S3DataDistributionType` field of the `S3DataSource` object that you pass to `CreateTrainingJob` to `FullyReplicated`. BlazingText takes care of distributing data across machines.

For the supervised text classification mode, a C5 instance is recommended if the training dataset is less than 2 GB. For larger datasets, use an instance with a single GPU (ml.p2.xlarge or ml.p3.2xlarge).

## BlazingText Sample Notebooks

For a sample notebook that uses the SageMaker BlazingText algorithm to train and deploy supervised binary and multiclass classification models, see [Blazing Text classification on the DBpedia dataset](#). For instructions for creating and accessing Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). After creating and opening a notebook instance, choose the **SageMaker Examples** tab to see a list of all the SageMaker examples. The topic modeling example notebooks that use the BlazingText are located in the **Introduction to Amazon algorithms** section. To open a notebook, choose its **Use** tab, then choose **Create copy**.

## BlazingText Hyperparameters

When you start a training job with a `CreateTrainingJob` request, you specify a training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The hyperparameters for the BlazingText algorithm depend on which mode you use: Word2Vec (unsupervised) and Text Classification (supervised).

### Word2Vec Hyperparameters

The following table lists the hyperparameters for the BlazingText Word2Vec training algorithm provided by Amazon SageMaker.

Parameter Name	Description
<code>mode</code>	<p>The Word2vec architecture used for training.</p> <p><b>Required</b></p> <p>Valid values: <code>batch_skipgram</code>, <code>skipgram</code>, or <code>cbow</code></p>
<code>batch_size</code>	<p>The size of each batch when <code>mode</code> is set to <code>batch_skipgram</code>. Set to a number between 10 and 20.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 11</p>
<code>buckets</code>	<p>The number of hash buckets to use for subwords.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 2000000</p>
<code>epochs</code>	<p>The number of complete passes through the training data.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 5</p>
<code>evaluation</code>	<p>Whether the trained model is evaluated using the <a href="#">WordSimilarity-353 Test</a>.</p> <p><b>Optional</b></p> <p>Valid values: (Boolean) <code>True</code> or <code>False</code></p>

Parameter Name	Description
	Default value: <code>True</code>
<code>learning_rate</code>	<p>The step size used for parameter updates.</p> <p><b>Optional</b></p> <p>Valid values: Positive float</p> <p>Default value: 0.05</p>
<code>min_char</code>	<p>The minimum number of characters to use for subwords/character n-grams.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 3</p>
<code>min_count</code>	<p>Words that appear less than <code>min_count</code> times are discarded.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative integer</p> <p>Default value: 5</p>
<code>max_char</code>	<p>The maximum number of characters to use for subwords/character n-grams</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 6</p>
<code>negative_samples</code>	<p>The number of negative samples for the negative sample sharing strategy.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 5</p>
<code>sampling_threshold</code>	<p>The threshold for the occurrence of words. Words that appear with higher frequency in the training data are randomly down-sampled.</p> <p><b>Optional</b></p> <p>Valid values: Positive fraction. The recommended range is (0, 1e-3]</p> <p>Default value: 0.0001</p>

Parameter Name	Description
<code>subwords</code>	<p>Whether to learn subword embeddings or not.</p> <p><b>Optional</b></p> <p>Valid values: (Boolean) <code>True</code> or <code>False</code></p> <p>Default value: <code>False</code></p>
<code>vector_dim</code>	<p>The dimension of the word vectors that the algorithm learns.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 100</p>
<code>window_size</code>	<p>The size of the context window. The context window is the number of words surrounding the target word used for training.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 5</p>

### Text Classification Hyperparameters

The following table lists the hyperparameters for the Text Classification training algorithm provided by Amazon SageMaker.

**Note**

Although some of the parameters are common between the Text Classification and Word2Vec modes, they might have different meanings depending on the context.

Parameter Name	Description
<code>mode</code>	<p>The training mode.</p> <p><b>Required</b></p> <p>Valid values: <code>supervised</code></p>
<code>buckets</code>	<p>The number of hash buckets to use for word n-grams.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 2000000</p>
<code>early_stopping</code>	<p>Whether to stop training if validation accuracy doesn't improve after a patience number of epochs.</p> <p><b>Optional</b></p> <p>Valid values: (Boolean) <code>True</code> or <code>False</code></p> <p>Default value: <code>False</code></p>

Parameter Name	Description
epochs	<p>The maximum number of complete passes through the training data.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 5</p>
learning_rate	<p>The step size used for parameter updates.</p> <p><b>Optional</b></p> <p>Valid values: Positive float</p> <p>Default value: 0.05</p>
min_count	<p>Words that appear less than <code>min_count</code> times are discarded.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative integer</p> <p>Default value: 5</p>
min_epochs	<p>The minimum number of epochs to train before early stopping logic is invoked.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 5</p>
patience	<p>The number of epochs to wait before applying early stopping when no progress is made on the validation set. Used only when <code>early_stopping</code> is <code>True</code>.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 4</p>
vector_dim	<p>The dimension of the embedding layer.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 100</p>
word_ngrams	<p>The number of word n-gram features to use.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 2</p>

## Tune a BlazingText Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics Computed by the BlazingText Algorithm

The BlazingText Word2Vec algorithm (skipgram, cbow, and batch\_skipgram modes) reports on a single metric during training: `train:mean_rho`. This metric is computed on [WS-353 word similarity datasets](#). When tuning the hyperparameter values for the Word2Vec algorithm, use this metric as the objective.

The BlazingText Text Classification algorithm (supervised mode), also reports on a single metric during training: the `validation:accuracy`. When tuning the hyperparameter values for the text classification algorithm, use these metrics as the objective.

Metric Name	Description	Optimization Direction
<code>train:mean_rho</code>	The mean rho (Spearman's rank correlation coefficient) on <a href="#">WS-353 word similarity datasets</a>	Maximize
<code>validation:accuracy</code>	The classification accuracy on the user-specified validation dataset	Maximize

### Tunable BlazingText Hyperparameters

#### Tunable Hyperparameters for the Word2Vec Algorithm

Tune an Amazon SageMaker BlazingText Word2Vec model with the following hyperparameters. The hyperparameters that have the greatest impact on Word2Vec objective metrics are: `mode`, `learning_rate`, `window_size`, `vector_dim`, and `negative_samples`.

Parameter Name	Parameter Type	Recommended Ranges or Values
<code>batch_size</code>	<code>IntegerParameterRange</code>	[8-32]
<code>epochs</code>	<code>IntegerParameterRange</code>	[5-15]
<code>learning_rate</code>	<code>ContinuousParameterRange</code>	MinValue: 0.005, MaxValue: 0.01
<code>min_count</code>	<code>IntegerParameterRange</code>	[0-100]
<code>mode</code>	<code>CategoricalParameterRange</code>	['batch_skipgram', 'skipgram', 'cbow']
<code>negative_samples</code>	<code>IntegerParameterRange</code>	[5-25]
<code>sampling_threshold</code>	<code>ContinuousParameterRange</code>	MinValue: 0.0001, MaxValue: 0.001
<code>vector_dim</code>	<code>IntegerParameterRange</code>	[32-300]

Parameter Name	Parameter Type	Recommended Ranges or Values
window_size	IntegerParameterRange	[1-10]

### Tunable Hyperparameters for the Text Classification Algorithm

Tune an Amazon SageMaker BlazingText text classification model with the following hyperparameters.

Parameter Name	Parameter Type	Recommended Ranges or Values
buckets	IntegerParameterRange	[1000000-10000000]
epochs	IntegerParameterRange	[5-15]
learning_rate	ContinuousParameterRange	MinValue: 0.005, MaxValue: 0.01
min_count	IntegerParameterRange	[0-100]
mode	CategoricalParameterRange	['supervised']
vector_dim	IntegerParameterRange	[32-300]
word_ngrams	IntegerParameterRange	[1-3]

## DeepAR Forecasting Algorithm

The Amazon SageMaker DeepAR forecasting algorithm is a supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNN). Classical forecasting methods, such as autoregressive integrated moving average (ARIMA) or exponential smoothing (ETS), fit a single model to each individual time series. They then use that model to extrapolate the time series into the future.

In many applications, however, you have many similar time series across a set of cross-sectional units. For example, you might have time series groupings for demand for different products, server loads, and requests for webpages. For this type of application, you can benefit from training a single model jointly over all of the time series. DeepAR takes this approach. When your dataset contains hundreds of related time series, DeepAR outperforms the standard ARIMA and ETS methods. You can also use the trained model to generate forecasts for new time series that are similar to the ones it has been trained on.

The training input for the DeepAR algorithm is one or, preferably, more target time series that have been generated by the same process or similar processes. Based on this input dataset, the algorithm trains a model that learns an approximation of this process/processes and uses it to predict how the target time series evolves. Each target time series can be optionally associated with a vector of static (time-independent) categorical features provided by the `cat` field and a vector of dynamic (time-dependent) time series provided by the `dynamic_feat` field. SageMaker trains the DeepAR model by randomly sampling training examples from each target time series in the training dataset. Each training example consists of a pair of adjacent context and prediction windows with fixed predefined lengths. To control how far in the past the network can see, use the `context_length` hyperparameter. To control how far in the future predictions can be made, use the `prediction_length` hyperparameter. For more information, see [How the DeepAR Algorithm Works \(p. 1378\)](#).

### Topics

- [Input/Output Interface for the DeepAR Algorithm \(p. 1375\)](#)

- [Best Practices for Using the DeepAR Algorithm \(p. 1377\)](#)
- [EC2 Instance Recommendations for the DeepAR Algorithm \(p. 1378\)](#)
- [DeepAR Sample Notebooks \(p. 1378\)](#)
- [How the DeepAR Algorithm Works \(p. 1378\)](#)
- [DeepAR Hyperparameters \(p. 1381\)](#)
- [Tune a DeepAR Model \(p. 1385\)](#)
- [DeepAR Inference Formats \(p. 1386\)](#)

## Input/Output Interface for the DeepAR Algorithm

DeepAR supports two data channels. The required `train` channel describes the training dataset. The optional `test` channel describes a dataset that the algorithm uses to evaluate model accuracy after training. You can provide training and test datasets in [JSON Lines](#) format. Files can be in gzip or [Parquet](#) file format.

When specifying the paths for the training and test data, you can specify a single file or a directory that contains multiple files, which can be stored in subdirectories. If you specify a directory, DeepAR uses all files in the directory as inputs for the corresponding channel, except those that start with a period (.) and those named `_SUCCESS`. This ensures that you can directly use output folders produced by Spark jobs as input channels for your DeepAR training jobs.

By default, the DeepAR model determines the input format from the file extension (`.json`, `.json.gz`, or `.parquet`) in the specified input path. If the path does not end in one of these extensions, you must explicitly specify the format in the SDK for Python. Use the `content_type` parameter of the [s3\\_input](#) class.

The records in your input files should contain the following fields:

- `start`—A string with the format `YYYY-MM-DD HH:MM:SS`. The start timestamp can't contain time zone information.
- `target`—An array of floating-point values or integers that represent the time series. You can encode missing values as `null` literals, or as "NaN" strings in JSON, or as `nan` floating-point values in Parquet.
- `dynamic_feat` (optional)—An array of arrays of floating-point values or integers that represents the vector of custom feature time series (dynamic features). If you set this field, all records must have the same number of inner arrays (the same number of feature time series). In addition, each inner array must be the same length as the associated `target` value plus length of the context. Missing values are not supported in the features. For example, if `target` time series represents the demand of different products, an associated `dynamic_feat` might be a boolean time-series which indicates whether a promotion was applied (1) to the particular product or not (0):

```
{"start": ..., "target": [1, 5, 10, 2], "dynamic_feat": [[0, 1, 1, 0]]}
```

- `cat` (optional)—An array of categorical features that can be used to encode the groups that the record belongs to. Categorical features must be encoded as a 0-based sequence of positive integers. For example, the categorical domain {R, G, B} can be encoded as {0, 1, 2}. All values from each categorical domain must be represented in the training dataset. That's because the DeepAR algorithm can forecast only for categories that have been observed during training. And, each categorical feature is embedded in a low-dimensional space whose dimensionality is controlled by the `embedding_dimension` hyperparameter. For more information, see [DeepAR Hyperparameters \(p. 1381\)](#).

If you use a JSON file, it must be in [JSON Lines](#) format. For example:

```
{"start": "2009-11-01 00:00:00", "target": [4.3, "NaN", 5.1, ...], "cat": [0, 1], "dynamic_feat": [[1.1, 1.2, 0.5, ...]]}
```

```
{"start": "2012-01-30 00:00:00", "target": [1.0, -5.0, ...], "cat": [2, 3], "dynamic_feat":  
[[1.1, 2.05, ...]]}  
{"start": "1999-01-30 00:00:00", "target": [2.0, 1.0], "cat": [1, 4], "dynamic_feat":  
[[1.3, 0.4]]}
```

In this example, each time series has two associated categorical features and one time series features.

For Parquet, you use the same three fields as columns. In addition, "start" can be the `datetime` type. You can compress Parquet files using gzip (`gzip`) or the Snappy compression library (`snappy`).

If the algorithm is trained without `cat` and `dynamic_feat` fields, it learns a "global" model, that is a model that is agnostic to the specific identity of the target time series at inference time and is conditioned only on its shape.

If the model is conditioned on the `cat` and `dynamic_feat` feature data provided for each time series, the prediction will probably be influenced by the character of time series with the corresponding `cat` features. For example, if the target time series represents the demand of clothing items, you can associate a two-dimensional `cat` vector that encodes the type of item (e.g. 0 = shoes, 1 = dress) in the first component and the color of an item (e.g. 0 = red, 1 = blue) in the second component. A sample input would look as follows:

```
{ "start": ..., "target": ..., "cat": [0, 0], ... } # red shoes  
{ "start": ..., "target": ..., "cat": [1, 1], ... } # blue dress
```

At inference time, you can request predictions for targets with `cat` values that are combinations of the `cat` values observed in the training data, for example:

```
{ "start": ..., "target": ..., "cat": [0, 1], ... } # blue shoes  
{ "start": ..., "target": ..., "cat": [1, 0], ... } # red dress
```

The following guidelines apply to training data:

- The start time and length of the time series can differ. For example, in marketing, products often enter a retail catalog at different dates, so their start dates naturally differ. But all series must have the same frequency, number of categorical features, and number of dynamic features.
- Shuffle the training file with respect to the position of the time series in the file. In other words, the time series should occur in random order in the file.
- Make sure to set the `start` field correctly. The algorithm uses the `start` timestamp to derive the internal features.
- If you use categorical features (`cat`), all time series must have the same number of categorical features. If the dataset contains the `cat` field, the algorithm uses it and extracts the cardinality of the groups from the dataset. By default, cardinality is "auto". If the dataset contains the `cat` field, but you don't want to use it, you can disable it by setting `cardinality` to "" . If a model was trained using a `cat` feature, you must include it for inference.
- If your dataset contains the `dynamic_feat` field, the algorithm uses it automatically. All time series have to have the same number of feature time series. The time points in each of the feature time series correspond one-to-one to the time points in the target. In addition, the entry in the `dynamic_feat` field should have the same length as the target. If the dataset contains the `dynamic_feat` field, but you don't want to use it, disable it by setting(`num_dynamic_feat` to "" ). If the model was trained with the `dynamic_feat` field, you must provide this field for inference. In addition, each of the features has to have the length of the provided target plus the `prediction_length`. In other words, you must provide the feature value in the future.

If you specify optional test channel data, the DeepAR algorithm evaluates the trained model with different accuracy metrics. The algorithm calculates the root mean square error (RMSE) over the test data as follows:

$$\text{RMSE} = \sqrt{\frac{1}{nT} \sum_{i,t} (\hat{y}_{i,t} - y_{i,t})^2}$$

$y_{i,t}$  is the true value of time series  $i$  at the time  $t$ .  $\hat{y}_{i,t}$  is the mean prediction. The sum is over all  $n$  time series in the test set and over the last  $T$  time points for each time series, where  $T$  corresponds to the forecast horizon. You specify the length of the forecast horizon by setting the [prediction\\_length](#) hyperparameter. For more information, see [DeepAR Hyperparameters \(p. 1381\)](#).

In addition, the algorithm evaluates the accuracy of the forecast distribution using weighted quantile loss. For a quantile in the range  $[0, 1]$ , the weighted quantile loss is defined as follows:

$$\text{wQuantileLoss}[\tau] = 2 \sum_{i,t} \frac{Q_{i,t}^{(\tau)}}{|y_{i,t}|}, \quad \text{with} \quad Q_{i,t}^{(\tau)} = \begin{cases} (1-\tau)|q_{i,t}^{(\tau)} - y_{i,t}| & \text{if } q_{i,t}^{(\tau)} > y_{i,t} \\ \tau|q_{i,t}^{(\tau)} - y_{i,t}| & \text{otherwise} \end{cases}$$

$q_{i,t}^{(\tau)}$  is the  $\tau$ -quantile of the distribution that the model predicts. To specify which quantiles to calculate loss for, set the [test\\_quantiles](#) hyperparameter. In addition to these, the average of the prescribed quantile losses is reported as part of the training logs. For information, see [DeepAR Hyperparameters \(p. 1381\)](#).

For inference, DeepAR accepts JSON format and the following fields:

- "instances", which includes one or more time series in JSON Lines format
- A name of "configuration", which includes parameters for generating the forecast

For more information, see [DeepAR Inference Formats \(p. 1386\)](#).

## Best Practices for Using the DeepAR Algorithm

When preparing your time series data, follow these best practices to achieve the best results:

- Except for when splitting your dataset for training and testing, always provide the entire time series for training, testing, and when calling the model for inference. Regardless of how you set `context_length`, don't break up the time series or provide only a part of it. The model uses data points further back than the value set in `context_length` for the lagged values feature.
- When tuning a DeepAR model, you can split the dataset to create a training dataset and a test dataset. In a typical evaluation, you would test the model on the same time series used for training, but on the future `prediction_length` time points that follow immediately after the last time point visible during training. You can create training and test datasets that satisfy this criteria by using the entire dataset (the full length of all time series that are available) as a test set and removing the last `prediction_length` points from each time series for training. During training, the model doesn't see the target values for time points on which it is evaluated during testing. During testing, the algorithm withholds the last `prediction_length` points of each time series in the test set and generates a prediction. Then it compares the forecast with the withheld values. You can create more complex evaluations by repeating time series multiple times in the test set, but cutting them at different endpoints. With this approach, accuracy metrics are averaged over multiple forecasts from different time points. For more information, see [Tune a DeepAR Model \(p. 1385\)](#).
- Avoid using very large values ( $>400$ ) for the `prediction_length` because it makes the model slow and less accurate. If you want to forecast further into the future, consider aggregating your data at a lower frequency. For example, use `5min` instead of `1min`.
- Because lags are used, a model can look further back in the time series than the value specified for `context_length`. Therefore, you don't need to set this parameter to a large value. We recommend starting with the value that you used for `prediction_length`.

- We recommend training a DeepAR model on as many time series as are available. Although a DeepAR model trained on a single time series might work well, standard forecasting algorithms, such as ARIMA or ETS, might provide more accurate results. The DeepAR algorithm starts to outperform the standard methods when your dataset contains hundreds of related time series. Currently, DeepAR requires that the total number of observations available across all training time series is at least 300.

## EC2 Instance Recommendations for the DeepAR Algorithm

You can train DeepAR on both GPU and CPU instances and in both single and multi-machine settings. We recommend starting with a single CPU instance (for example, ml.c4.2xlarge or ml.c4.4xlarge), and switching to GPU instances and multiple machines only when necessary. Using GPUs and multiple machines improves throughput only for larger models (with many cells per layer and many layers) and for large mini-batch sizes (for example, greater than 512).

For inference, DeepAR supports only CPU instances.

Specifying large values for `context_length`, `prediction_length`, `num_cells`, `num_layers`, or `mini_batch_size` can create models that are too large for small instances. In this case, use a larger instance type or reduce the values for these parameters. This problem also frequently occurs when running hyperparameter tuning jobs. In that case, use an instance type large enough for the model tuning job and consider limiting the upper values of the critical parameters to avoid job failures.

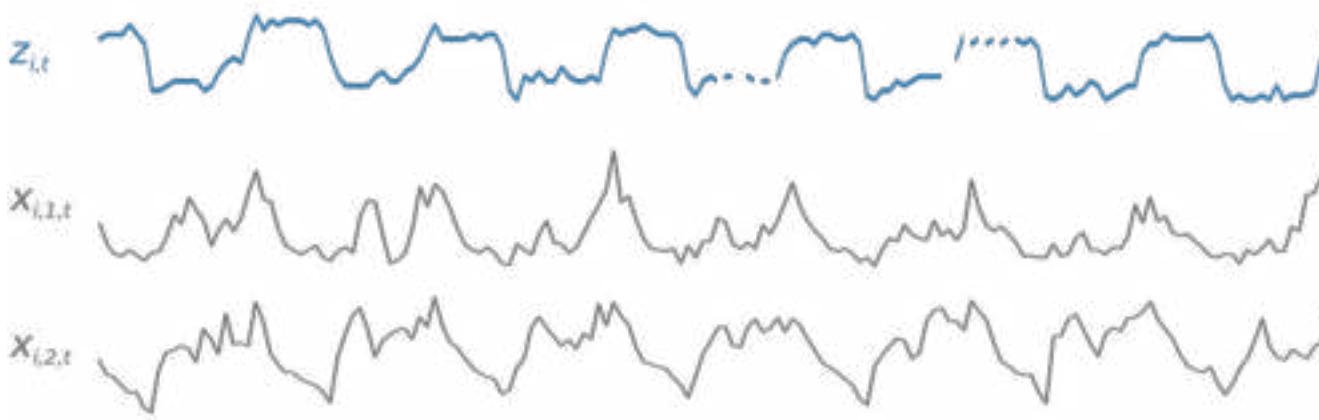
## DeepAR Sample Notebooks

For a sample notebook that shows how to prepare a time series dataset for training the SageMaker DeepAR algorithm and how to deploy the trained model for performing inferences, see [Time series forecasting with DeepAR - Synthetic data](#) as well as [DeepAR demo on electricity dataset](#), which illustrates the advanced features of DeepAR on a real world dataset. For instructions on creating and accessing Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). After creating and opening a notebook instance, choose the **SageMaker Examples** tab to see a list of all of the SageMaker examples. To open a notebook, choose its **Use** tab, and choose **Create copy**.

## How the DeepAR Algorithm Works

During training, DeepAR accepts a training dataset and an optional test dataset. It uses the test dataset to evaluate the trained model. In general, the datasets don't have to contain the same set of time series. You can use a model trained on a given training set to generate forecasts for the future of the time series in the training set, and for other time series. Both the training and the test datasets consist of one or, preferably, more target time series. Each target time series can optionally be associated with a vector of feature time series and a vector of categorical features. For more information, see [Input/Output Interface for the DeepAR Algorithm \(p. 1375\)](#).

For example, the following is an element of a training set indexed by  $i$  which consists of a target time series,  $Z_{i,t}$ , and two associated feature time series,  $X_{i,1,t}$  and  $X_{i,2,t}$ :

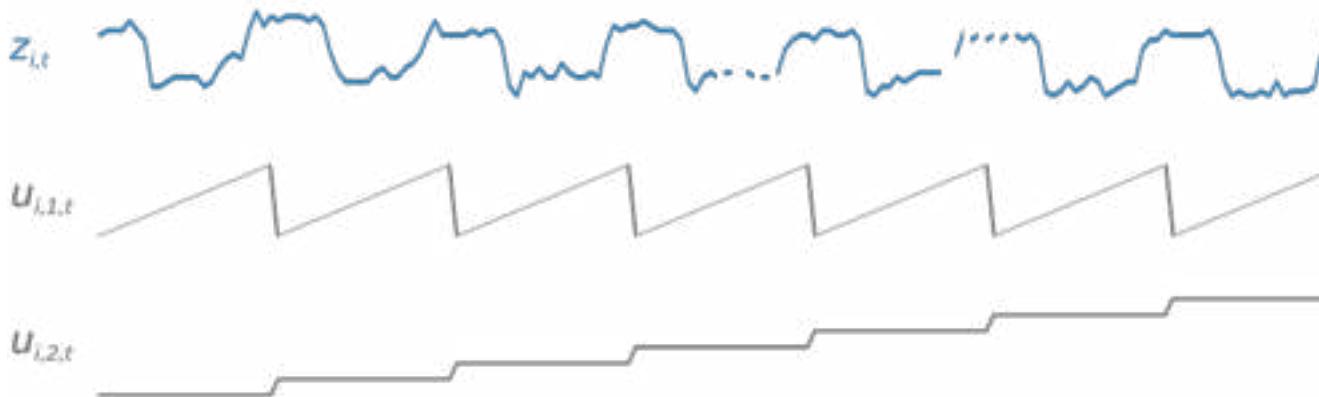


The target time series might contain missing values, which are represented by line breaks in the time series. DeepAR supports only feature time series that are known in the future. This allows you to run "what if?" scenarios. What happens, for example, if I change the price of a product in some way?

Each target time series can also be associated with a number of categorical features. You can use these features to encode which groupings a time series belongs to. Categorical features allow the model to learn typical behavior for groups, which it can use to increase model accuracy. DeepAR implements this by learning an embedding vector for each group that captures the common properties of all time series in the group.

#### How Feature Time Series Work in the DeepAR Algorithm

To facilitate learning time-dependent patterns, such as spikes during weekends, DeepAR automatically creates feature time series based on the frequency of the target time series. For example, DeepAR creates two feature time series (day of the month and day of the year) for a weekly time series frequency. It uses these derived feature time series with the custom feature time series that you provide during training and inference. The following figure shows two of these derived time series features:  $u_{i,1,t}$  represents the hour of the day and  $u_{i,2,t}$  the day of the week.

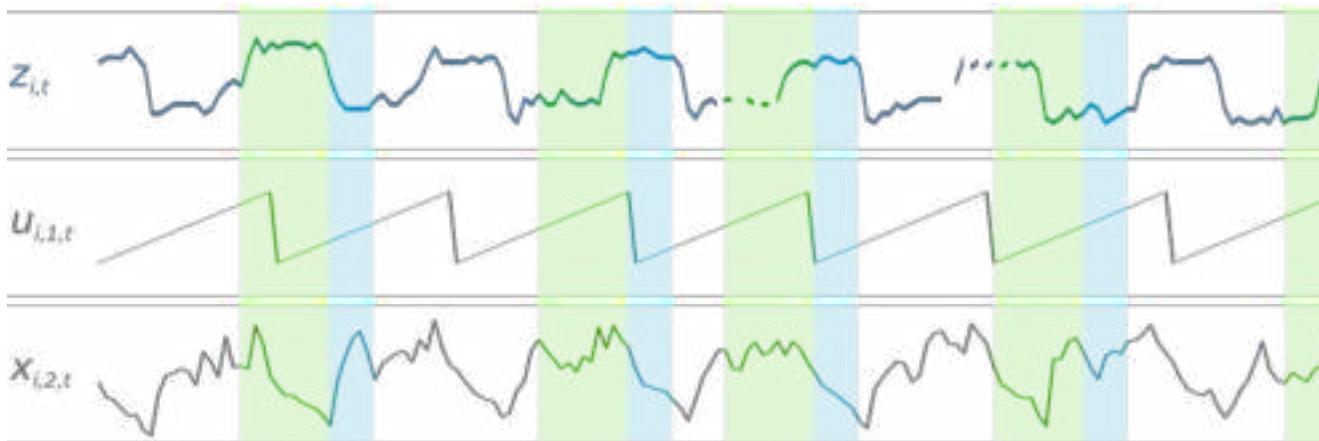


The DeepAR algorithm automatically generates these feature time series. The following table lists the derived features for the supported basic time frequencies.

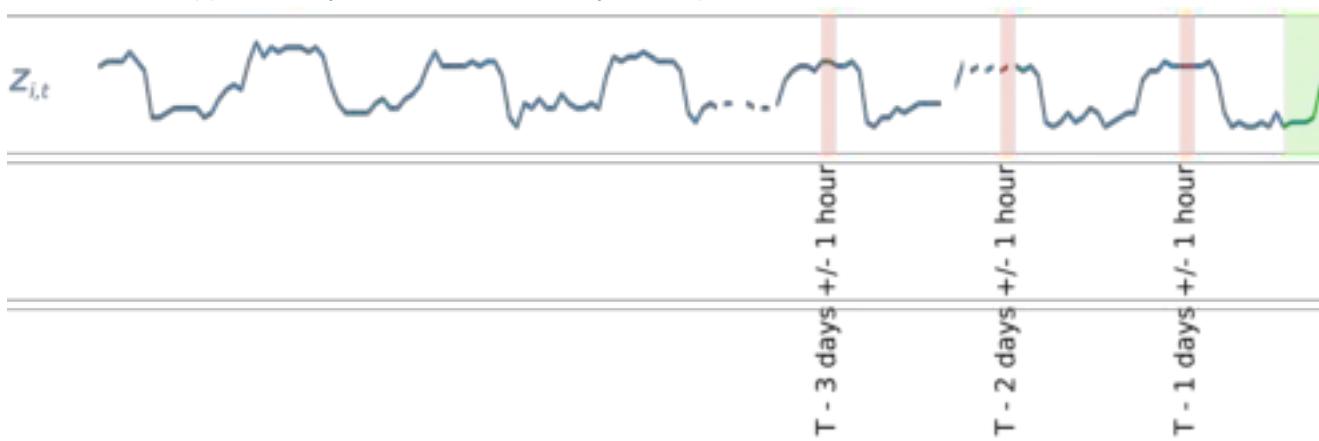
Frequency of the Time Series	Derived Features
Minute	minute-of-hour, hour-of-day, day-of-week, day-of-month, day-of-year

Frequency of the Time Series	Derived Features
Hour	hour-of-day, day-of-week, day-of-month, day-of-year
Day	day-of-week, day-of-month, day-of-year
Week	day-of-month, week-of-year
Month	month-of-year

DeepAR trains a model by randomly sampling several training examples from each of the time series in the training dataset. Each training example consists of a pair of adjacent context and prediction windows with fixed predefined lengths. The `context_length` hyperparameter controls how far in the past the network can see, and the `prediction_length` hyperparameter controls how far in the future predictions can be made. During training, the algorithm ignores training set elements containing time series that are shorter than a specified prediction length. The following figure represents five samples with context lengths of 12 hours and prediction lengths of 6 hours drawn from element  $i$ . For brevity, we've omitted the feature time series  $x_{i,1,t}$  and  $u_{i,2,t}$ .



To capture seasonality patterns, DeepAR also automatically feeds lagged values from the target time series. In the example with hourly frequency, for each time index,  $t = T$ , the model exposes the  $z_{i,t}$  values, which occurred approximately one, two, and three days in the past.



For inference, the trained model takes as input target time series, which might or might not have been used during training, and forecasts a probability distribution for the next `prediction_length` values.

Because DeepAR is trained on the entire dataset, the forecast takes into account patterns learned from similar time series.

For information on the mathematics behind DeepAR, see [DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks](#).

## DeepAR Hyperparameters

Parameter Name	Description
context_length	<p>The number of time-points that the model gets to see before making the prediction. The value for this parameter should be about the same as the <code>prediction_length</code>. The model also receives lagged inputs from the target, so <code>context_length</code> can be much smaller than typical seasonalities. For example, a daily time series can have yearly seasonality. The model automatically includes a lag of one year, so the context length can be shorter than a year. The lag values that the model picks depend on the frequency of the time series. For example, lag values for daily frequency are previous week, 2 weeks, 3 weeks, 4 weeks, and year.</p> <p><b>Required</b></p> <p>Valid values: Positive integer</p>
epochs	<p>The maximum number of passes over the training data. The optimal value depends on your data size and learning rate. See also <code>early_stopping_patience</code>. Typical values range from 10 to 1000.</p> <p><b>Required</b></p> <p>Valid values: Positive integer</p>
prediction_length	<p>The number of time-steps that the model is trained to predict, also called the forecast horizon. The trained model always generates forecasts with this length. It can't generate longer forecasts. The <code>prediction_length</code> is fixed when a model is trained and it cannot be changed later.</p> <p><b>Required</b></p> <p>Valid values: Positive integer</p>
time_freq	<p>The granularity of the time series in the dataset. Use <code>time_freq</code> to select appropriate date features and lags. The model supports the following basic frequencies. It also supports multiples of these basic frequencies. For example, <code>5min</code> specifies a frequency of 5 minutes.</p> <ul style="list-style-type: none"> <li>• <code>M</code>: monthly</li> <li>• <code>W</code>: weekly</li> <li>• <code>D</code>: daily</li> <li>• <code>H</code>: hourly</li> <li>• <code>min</code>: every minute</li> </ul> <p><b>Required</b></p>

Parameter Name	Description
	<p>Valid values: An integer followed by <i>M</i>, <i>W</i>, <i>D</i>, <i>H</i>, or <i>min</i>. For example, 5min.</p>
<code>cardinality</code>	<p>When using the categorical features (<code>cat</code>), <code>cardinality</code> is an array specifying the number of categories (groups) per categorical feature. Set this to <code>auto</code> to infer the cardinality from the data. The <code>auto</code> mode also works when no categorical features are used in the dataset. This is the recommended setting for the parameter.</p> <p>Set <code>cardinality</code> to <code>ignore</code> to force DeepAR to not use categorical features, even if they are present in the data.</p> <p>To perform additional data validation, it is possible to explicitly set this parameter to the actual value. For example, if two categorical features are provided where the first has 2 and the other has 3 possible values, set this to [2, 3].</p> <p>For more information on how to use categorical feature, see the data-section on the main documentation page of DeepAR.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code>, <code>ignore</code>, array of positive integers, empty string, or</p> <p>Default value: <code>auto</code></p>
<code>dropout_rate</code>	<p>The dropout rate to use during training. The model uses zoneout regularization. For each iteration, a random subset of hidden neurons are not updated. Typical values are less than 0.2.</p> <p><b>Optional</b></p> <p>Valid values: float</p> <p>Default value: 0.1</p>
<code>early_stopping_patience</code>	<p>If this parameter is set, training stops when no progress is made within the specified number of epochs. The model that has the lowest loss is returned as the final model.</p> <p><b>Optional</b></p> <p>Valid values: integer</p>

Parameter Name	Description
embedding_dimension	<p>Size of embedding vector learned per categorical feature (same value is used for all categorical features).</p> <p>The DeepAR model can learn group-level time series patterns when a categorical grouping feature is provided. To do this, the model learns an embedding vector of size <code>embedding_dimension</code> for each group, capturing the common properties of all time series in the group. A larger <code>embedding_dimension</code> allows the model to capture more complex patterns. However, because increasing the <code>embedding_dimension</code> increases the number of parameters in the model, more training data is required to accurately learn these parameters. Typical values for this parameter are between 10-100.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 10</p>
learning_rate	<p>The learning rate used in training. Typical values range from 1e-4 to 1e-1.</p> <p><b>Optional</b></p> <p>Valid values: float</p> <p>Default value: 1e-3</p>
likelihood	<p>The model generates a probabilistic forecast, and can provide quantiles of the distribution and return samples. Depending on your data, select an appropriate likelihood (noise model) that is used for uncertainty estimates. The following likelihoods can be selected:</p> <ul style="list-style-type: none"> <li>• <i>gaussian</i>: Use for real-valued data.</li> <li>• <i>beta</i>: Use for real-valued targets between 0 and 1 inclusive.</li> <li>• <i>negative-binomial</i>: Use for count data (non-negative integers).</li> <li>• <i>student-T</i>: An alternative for real-valued data that works well for bursty data.</li> <li>• <i>deterministic-L1</i>: A loss function that does not estimate uncertainty and only learns a point forecast.</li> </ul> <p><b>Optional</b></p> <p>Valid values: One of <i>gaussian</i>, <i>beta</i>, <i>negative-binomial</i>, <i>student-T</i>, or <i>deterministic-L1</i>.</p> <p>Default value: <code>student-T</code></p>

Parameter Name	Description
mini_batch_size	<p>The size of mini-batches used during training. Typical values range from 32 to 512.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 128</p>
num_cells	<p>The number of cells to use in each hidden layer of the RNN. Typical values range from 30 to 100.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 40</p>
num_dynamic_feat	<p>The number of <code>dynamic_feat</code> provided in the data. Set this to <code>auto</code> to infer the number of dynamic features from the data. The <code>auto</code> mode also works when no dynamic features are used in the dataset. This is the recommended setting for the parameter.</p> <p>To force DeepAR to not use dynamic features, even if they are present in the data, set <code>num_dynamic_feat</code> to <code>ignore</code>.</p> <p>To perform additional data validation, it is possible to explicitly set this parameter to the actual integer value. For example, if two dynamic features are provided, set this to 2.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code>, <code>ignore</code>, positive integer, or empty string</p> <p>Default value: <code>auto</code></p>
num_eval_samples	<p>The number of samples that are used per time-series when calculating test accuracy metrics. This parameter does not have any influence on the training or the final model. In particular, the model can be queried with a different number of samples. This parameter only affects the reported accuracy scores on the test channel after training. Smaller values result in faster evaluation, but then the evaluation scores are typically worse and more uncertain. When evaluating with higher quantiles, for example 0.95, it may be important to increase the number of evaluation samples.</p> <p><b>Optional</b></p> <p>Valid values: integer</p> <p>Default value: 100</p>

Parameter Name	Description
<code>num_layers</code>	<p>The number of hidden layers in the RNN. Typical values range from 1 to 4.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 2</p>
<code>test_quantiles</code>	<p>Quantiles for which to calculate quantile loss on the test channel.</p> <p><b>Optional</b></p> <p>Valid values: array of floats</p> <p>Default value: [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]</p>

## Tune a DeepAR Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics Computed by the DeepAR Algorithm

The DeepAR algorithm reports three metrics, which are computed during training. When tuning a model, choose one of these as the objective. For the objective, use either the forecast accuracy on a provided test channel (recommended) or the training loss. For recommendations for the training/test split for the DeepAR algorithm, see [Best Practices for Using the DeepAR Algorithm \(p. 1377\)](#).

Metric Name	Description	Optimization Direction
<code>test:RMSE</code>	The root mean square error between the forecast and the actual target computed on the test set.	Minimize
<code>test:mean_wQuantileLoss</code>	The average overall quantile losses computed on the test set. To control which quantiles are used, set the <code>test_quantiles</code> hyperparameter.	Minimize
<code>train:final_loss</code>	The training negative log-likelihood loss averaged over the last training epoch for the model.	Minimize

### Tunable Hyperparameters for the DeepAR Algorithm

Tune a DeepAR model with the following hyperparameters. The hyperparameters that have the greatest impact, listed in order from the most to least impactful, on DeepAR objective metrics are: `epochs`, `context_length`, `mini_batch_size`, `learning_rate`, and `num_cells`.

Parameter Name	Parameter Type	Recommended Ranges
mini_batch_size	IntegerParameterRanges	MinValue: 32, MaxValue: 1028
epochs	IntegerParameterRanges	MinValue: 1, MaxValue: 1000
context_length	IntegerParameterRanges	MinValue: 1, MaxValue: 200
num_cells	IntegerParameterRanges	MinValue: 30, MaxValue: 200
num_layers	IntegerParameterRanges	MinValue: 1, MaxValue: 8
dropout_rate	ContinuousParameterRange	MinValue: 0.00, MaxValue: 0.2
embedding_dimension	IntegerParameterRanges	MinValue: 1, MaxValue: 50
learning_rate	ContinuousParameterRange	MinValue: 1e-5, MaxValue: 1e-1

## DeepAR Inference Formats

### DeepAR JSON Request Formats

Query a trained model by using the model's endpoint. The endpoint takes the following JSON request format.

In the request, the `instances` field corresponds to the time series that should be forecast by the model.

If the model was trained with categories, you must provide a `cat` for each instance. If the model was trained without the `cat` field, it should be omitted.

If the model was trained with a custom feature time series (`dynamic_feat`), you have to provide the same number of `dynamic_feat` values for each instance. Each of them should have a length given by `length(target) + prediction_length`, where the last `prediction_length` values correspond to the time points in the future that will be predicted. If the model was trained without custom feature time series, the field should not be included in the request.

```
{
  "instances": [
    {
      "start": "2009-11-01 00:00:00",
      "target": [4.0, 10.0, "NaN", 100.0, 113.0],
      "cat": [0, 1],
      "dynamic_feat": [[1.0, 1.1, 2.1, 0.5, 3.1, 4.1, 1.2, 5.0, ...]]
    },
    {
      "start": "2012-01-30",
      "target": [1.0],
      "cat": [2, 1],
      "dynamic_feat": [[2.0, 3.1, 4.5, 1.5, 1.8, 3.2, 0.1, 3.0, ...]]
    },
    {
      "start": "1999-01-30",
      ...
    }
  ]
}
```

```

        "target": [2.0, 1.0],
        "cat": [1, 3],
        "dynamic_feat": [[1.0, 0.1, -2.5, 0.3, 2.0, -1.2, -0.1, -3.0, ...]]
    }
],
"configuration": {
    "num_samples": 50,
    "output_types": ["mean", "quantiles", "samples"],
    "quantiles": ["0.5", "0.9"]
}
}

```

The `configuration` field is optional. `configuration.num_samples` sets the number of sample paths that the model generates to estimate the mean and quantiles. `configuration.output_types` describes the information that will be returned in the request. Valid values are `"mean"`, `"quantiles"` and `"samples"`. If you specify `"quantiles"`, each of the quantile values in `configuration.quantiles` is returned as a time series. If you specify `"samples"`, the model also returns the raw samples used to calculate the other outputs.

### DeepAR JSON Response Formats

The following is the format of a response, where `[ ... ]` are arrays of numbers:

```
{
  "predictions": [
    {
      "quantiles": {
        "0.9": [...],
        "0.5": [...]
      },
      "samples": [...],
      "mean": [...]
    },
    {
      "quantiles": {
        "0.9": [...],
        "0.5": [...]
      },
      "samples": [...],
      "mean": [...]
    },
    {
      "quantiles": {
        "0.9": [...],
        "0.5": [...]
      },
      "samples": [...],
      "mean": [...]
    }
  ]
}
```

DeepAR has a response timeout of 60 seconds. When passing multiple time series in a single request, the forecasts are generated sequentially. Because the forecast for each time series typically takes about 300 to 1000 milliseconds or longer, depending on the model size, passing too many time series in a single request can cause time outs. It's better to send fewer time series per request and send more requests. Because the DeepAR algorithm uses multiple workers per instance, you can achieve much higher throughput by sending multiple requests in parallel.

By default, DeepAR uses one worker per CPU for inference, if there is sufficient memory per CPU. If the model is large and there isn't enough memory to run a model on each CPU, the number of workers is reduced. The number of workers used for inference can be overwritten using the environment variable

`MODEL_SERVER_WORKERS` For example, by setting `MODEL_SERVER_WORKERS=1`) when calling the SageMaker [CreateModel](#) API.

### Batch Transform with the DeepAR Algorithm

DeepAR forecasting supports getting inferences by using batch transform from data using the JSON Lines format. In this format, each record is represented on a single line as a JSON object, and lines are separated by newline characters. The format is identical to the JSON Lines format used for model training. For information, see [Input/Output Interface for the DeepAR Algorithm \(p. 1375\)](#). For example:

```
{"start": "2009-11-01 00:00:00", "target": [4.3, "NaN", 5.1, ...], "cat": [0, 1],  
"dynamic_feat": [[1.1, 1.2, 0.5, ...]]}  
{ "start": "2012-01-30 00:00:00", "target": [1.0, -5.0, ...], "cat": [2, 3], "dynamic_feat":  
[[1.1, 2.05, ...]]}  
{ "start": "1999-01-30 00:00:00", "target": [2.0, 1.0], "cat": [1, 4], "dynamic_feat":  
[[1.3, 0.4]]}
```

#### Note

When creating the transformation job with [CreateTransformJob](#), set the `BatchStrategy` value to `SingleRecord` and set the `SplitType` value in the `TransformInput` configuration to `Line`, as the default values currently cause runtime failures.

Similar to the hosted endpoint inference request format, the `cat` and the `dynamic_feat` fields for each instance are required if both of the following are true:

- The model is trained on a dataset that contained both the `cat` and the `dynamic_feat` fields.
- The corresponding `cardinality` and `num_dynamic_feat` values used in the training job are not set to `" "`.

Unlike hosted endpoint inference, the configuration field is set once for the entire batch inference job using an environment variable named `DEEPAR_INFERENCE_CONFIG`. The value of `DEEPAR_INFERENCE_CONFIG` can be passed when the model is created by calling [CreateTransformJob](#) API. If `DEEPAR_INFERENCE_CONFIG` is missing in the container environment, the inference container uses the following default:

```
{  
    "num_samples": 100,  
    "output_types": ["mean", "quantiles"],  
    "quantiles": ["0.1", "0.2", "0.3", "0.4", "0.5", "0.6", "0.7", "0.8", "0.9"]  
}
```

The output is also in JSON Lines format, with one line per prediction, in an order identical to the instance order in the corresponding input file. Predictions are encoded as objects identical to the ones returned by responses in online inference mode. For example:

```
{ "quantiles": { "0.1": [...], "0.2": [...] }, "samples": [...], "mean": [...] }
```

Note that in the `TransformInput` configuration of the SageMaker [CreateTransformJob](#) request clients must explicitly set the `AssembleWith` value to `Line`, as the default value `None` concatenates all JSON objects on the same line.

For example, here is a SageMaker [CreateTransformJob](#) request for a DeepAR job with a custom `DEEPAR_INFERENCE_CONFIG`:

```
{  
    "BatchStrategy": "SingleRecord",  
    "Environment": {  
        "DEEPAR_INFERENCE_CONFIG" : "{ \"num_samples\": 200, \"output_types\": [\"mean\"] }",  
    },  
    "ModelArn": "arn:aws:sagemaker:us-west-2:123456789012:transformer:my-deepar-transformer",  
    "TransformInput": {  
        "FileContentLocation": "s3://my-bucket/input-data",  
        "AssembleWith": "Line",  
        "ContentType": "text/plain",  
        "SplitType": "Line",  
        "SourceType": "File",  
        "CompressionType": "None"  
    },  
    "TransformOutput": {  
        "FileContentLocation": "s3://my-bucket/output-data",  
        "AssembleWith": "Line",  
        "ContentType": "text/plain",  
        "SplitType": "Line",  
        "SourceType": "File",  
        "CompressionType": "None"  
    },  
    "TransformResources": {  
        "ClusterConfig": {  
            "InstanceType": "ml.m5.xlarge",  
            "InstanceCount": 1  
        }  
    },  
    "TransformTimeout": 3600  
}
```

```
    ...
},
"TransformInput": {
    "SplitType": "Line",
    ...
},
"TransformOutput": {
    "AssembleWith": "Line",
    ...
},
...
}
```

## Factorization Machines Algorithm

The Factorization Machines algorithm is a general-purpose supervised learning algorithm that you can use for both classification and regression tasks. It is an extension of a linear model that is designed to capture interactions between features within high dimensional sparse datasets economically. For example, in a click prediction system, the Factorization Machines model can capture click rate patterns observed when ads from a certain ad-category are placed on pages from a certain page-category. Factorization machines are a good choice for tasks dealing with high dimensional sparse datasets, such as click prediction and item recommendation.

### Note

The Amazon SageMaker implementation of the Factorization Machines algorithm considers only pair-wise (2nd order) interactions between features.

### Topics

- [Input/Output Interface for the Factorization Machines Algorithm \(p. 1389\)](#)
- [EC2 Instance Recommendation for the Factorization Machines Algorithm \(p. 1390\)](#)
- [Factorization Machines Sample Notebooks \(p. 1390\)](#)
- [How Factorization Machines Work \(p. 1390\)](#)
- [Factorization Machines Hyperparameters \(p. 1391\)](#)
- [Tune a Factorization Machines Model \(p. 1396\)](#)
- [Factorization Machines Response Formats \(p. 1397\)](#)

## Input/Output Interface for the Factorization Machines Algorithm

The Factorization Machines algorithm can be run in either in binary classification mode or regression mode. In each mode, a dataset can be provided to the **test** channel along with the train channel dataset. The scoring depends on the mode used. In regression mode, the testing dataset is scored using Root Mean Square Error (RMSE). In binary classification mode, the test dataset is scored using Binary Cross Entropy (Log Loss), Accuracy (at threshold=0.5) and F1 Score (at threshold =0.5).

For **training**, the Factorization Machines algorithm currently supports only the `recordIO-protobuf` format with `Float32` tensors. Because their use case is predominantly on sparse data, CSV is not a good candidate. Both File and Pipe mode training are supported for recordIO-wrapped protobuf.

For **inference**, the Factorization Machines algorithm supports the `application/json` and `x-recordio-protobuf` formats.

- For the **binary classification** problem, the algorithm predicts a score and a label. The label is a number and can be either 0 or 1. The score is a number that indicates how strongly the algorithm believes that the label should be 1. The algorithm computes score first and then derives the label from the score value. If the score is greater than or equal to 0.5, the label is 1.
- For the **regression** problem, just a score is returned and it is the predicted value. For example, if Factorization Machines is used to predict a movie rating, score is the predicted rating value.

Please see [Factorization Machines Sample Notebooks \(p. 1390\)](#) for more details on training and inference file formats.

## EC2 Instance Recommendation for the Factorization Machines Algorithm

The Amazon SageMaker Factorization Machines algorithm is highly scalable and can train across distributed instances. We recommend training and inference with CPU instances for both sparse and dense datasets. In some circumstances, training with one or more GPUs on dense data might provide some benefit. Training with GPUs is available only on dense data. Use CPU instances for sparse data.

## Factorization Machines Sample Notebooks

For a sample notebook that uses the SageMaker Factorization Machines algorithm to analyze the images of handwritten digits from zero to nine in the MNIST dataset, see [An Introduction to Factorization Machines with MNIST](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the SageMaker samples. Example notebooks that use Factorization Machines algorithm are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

## How Factorization Machines Work

The prediction task for a Factorization Machines model is to estimate a function  $\hat{y}$  from a feature set  $x_i$  to a target domain. This domain is real-valued for regression and binary for classification. The Factorization Machines model is supervised and so has a training dataset  $(x_i, y_j)$  available. The advantages this model presents lie in the way it uses a factorized parametrization to capture the pairwise feature interactions. It can be represented mathematically as follows:

$$\hat{y} = w_0 + \sum_i w_i x_i + \sum_i \sum_{j > i} \langle v_i, v_j \rangle x_i x_j$$

The three terms in this equation correspond respectively to the three components of the model:

- The  $w_0$  term represents the global bias.
- The  $w_i$  linear terms model the strength of the  $i^{\text{th}}$  variable.
- The  $\langle v_i, v_j \rangle$  factorization terms model the pairwise interaction between the  $i^{\text{th}}$  and  $j^{\text{th}}$  variable.

The global bias and linear terms are the same as in a linear model. The pairwise feature interactions are modeled in the third term as the inner product of the corresponding factors learned for each feature. Learned factors can also be considered as embedding vectors for each feature. For example, in a classification task, if a pair of features tends to co-occur more often in positive labeled samples, then the inner product of their factors would be large. In other words, their embedding vectors would be close to each other in cosine similarity. For more information about the Factorization Machines model, see [Factorization Machines](#).

For regression tasks, the model is trained by minimizing the squared error between the model prediction  $\hat{y}_n$  and the target value  $y_n$ . This is known as the square loss:

$$L = \frac{1}{N} \sum_n (y_n - \hat{y}_n)^2$$

For a classification task, the model is trained by minimizing the cross entropy loss, also known as the log loss:

$$L = \frac{1}{N} \sum_n [y_n \log \hat{p}_n + (1 - y_n) \log (1 - \hat{p}_n)]$$

where:

$$\hat{p}_n = \frac{1}{1 + e^{-\hat{y}_n}}$$

For more information about loss functions for classification, see [Loss functions for classification](#).

## Factorization Machines Hyperparameters

The following table contains the hyperparameters for the Factorization Machines algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order.

Parameter Name	Description
<code>feature_dim</code>	<p>The dimension of the input feature space. This could be very high with sparse input.</p> <p><b>Required</b></p> <p>Valid values: Positive integer. Suggested value range: [10000,10000000]</p>
<code>num_factors</code>	<p>The dimensionality of factorization.</p> <p><b>Required</b></p> <p>Valid values: Positive integer. Suggested value range: [2,1000], 64 typically generates good outcomes and is a good starting point.</p>
<code>predictor_type</code>	<p>The type of predictor.</p> <ul style="list-style-type: none"> <li>• <code>binary_classifier</code>: For binary classification tasks.</li> <li>• <code>regressor</code>: For regression tasks.</li> </ul> <p><b>Required</b></p> <p>Valid values: String: <code>binary_classifier</code> or <code>regressor</code></p>
<code>bias_init_method</code>	<p>The initialization method for the bias term:</p> <ul style="list-style-type: none"> <li>• <code>normal</code>: Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>bias_init_sigma</code>.</li> <li>• <code>uniform</code>: Initializes weights with random values uniformly sampled from a range specified by <code>-bias_init_scale</code>, <code>+bias_init_scale</code>.</li> <li>• <code>constant</code>: Initializes the weights to a scalar value specified by <code>bias_init_value</code>.</li> </ul> <p><b>Optional</b></p> <p>Valid values: <code>uniform</code>, <code>normal</code>, or <code>constant</code></p> <p>Default value: <code>normal</code></p>

Parameter Name	Description
bias_init_scale	<p>Range for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>uniform</code>.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
bias_init_sigma	<p>The standard deviation for initialization of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>normal</code>.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.01</p>
bias_init_value	<p>The initial value of the bias term. Takes effect if <code>bias_init_method</code> is set to <code>constant</code>.</p> <p><b>Optional</b></p> <p>Valid values: Float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
bias_lr	<p>The learning rate for the bias term.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.1</p>
bias_wd	<p>The weight decay for the bias term.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.01</p>
clip_gradient	<p>Gradient clipping optimizer parameter. Clips the gradient by projecting onto the interval [-<code>clip_gradient</code>, +<code>clip_gradient</code>].</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default value: None</p>

Parameter Name	Description
epochs	<p>The number of training epochs to run.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 1</p>
eps	<p>Epsilon parameter to avoid division by 0.</p> <p><b>Optional</b></p> <p>Valid values: Float. Suggested value: small.</p> <p>Default value: None</p>
factors_init_method	<p>The initialization method for factorization terms:</p> <ul style="list-style-type: none"> <li>• <code>normal</code>: Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>factors_init_sigma</code>.</li> <li>• <code>uniform</code>: Initializes weights with random values uniformly sampled from a range specified by <code>[-factors_init_scale, +factors_init_scale]</code>.</li> <li>• <code>constant</code>: Initializes the weights to a scalar value specified by <code>factors_init_value</code>.</li> </ul> <p><b>Optional</b></p> <p>Valid values: <code>uniform</code>, <code>normal</code>, or <code>constant</code>.</p> <p>Default value: <code>normal</code></p>
factors_init_scale	<p>The range for initialization of factorization terms. Takes effect if <code>factors_init_method</code> is set to <code>uniform</code>.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
factors_init_sigma	<p>The standard deviation for initialization of factorization terms. Takes effect if <code>factors_init_method</code> is set to <code>normal</code>.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.001</p>

Parameter Name	Description
<code>factors_init_value</code>	<p>The initial value of factorization terms. Takes effect if <code>factors_init_method</code> is set to <code>constant</code>.</p> <p><b>Optional</b></p> <p>Valid values: Float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>factors_lr</code>	<p>The learning rate for factorization terms.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.0001</p>
<code>factors_wd</code>	<p>The weight decay for factorization terms.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.00001</p>
<code>linear_lr</code>	<p>The learning rate for linear terms.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.001</p>
<code>linear_init_method</code>	<p>The initialization method for linear terms:</p> <ul style="list-style-type: none"> <li><code>normal</code> Initializes weights with random values sampled from a normal distribution with a mean of zero and standard deviation specified by <code>linear_init_sigma</code>.</li> <li><code>uniform</code> Initializes weights with random values uniformly sampled from a range specified by <code>[-linear_init_scale, +linear_init_scale]</code>.</li> <li><code>constant</code> Initializes the weights to a scalar value specified by <code>linear_init_value</code>.</li> </ul> <p><b>Optional</b></p> <p>Valid values: <code>uniform</code>, <code>normal</code>, or <code>constant</code>.</p> <p>Default value: <code>normal</code></p>

Parameter Name	Description
<code>linear_init_scale</code>	<p>Range for initialization of linear terms. Takes effect if <code>linear_init_method</code> is set to <code>uniform</code>.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>linear_init_sigma</code>	<p>The standard deviation for initialization of linear terms. Takes effect if <code>linear_init_method</code> is set to <code>normal</code>.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.01</p>
<code>linear_init_value</code>	<p>The initial value of linear terms. Takes effect if <code>linear_init_method</code> is set to <code>constant</code>.</p> <p><b>Optional</b></p> <p>Valid values: Float. Suggested value range: [1e-8, 512].</p> <p>Default value: None</p>
<code>linear_wd</code>	<p>The weight decay for linear terms.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative float. Suggested value range: [1e-8, 512].</p> <p>Default value: 0.001</p>
<code>mini_batch_size</code>	<p>The size of mini-batch used for training.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 1000</p>
<code>rescale_grad</code>	<p>Gradient rescaling optimizer parameter. If set, multiplies the gradient with <code>rescale_grad</code> before updating. Often choose to be 1.0/batch_size.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default value: None</p>

## Tune a Factorization Machines Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics Computed by the Factorization Machines Algorithm

The Factorization Machines algorithm has both binary classification and regression predictor types. The predictor type determines which metric you can use for automatic model tuning. The algorithm reports a `test:rmse` regressor metric, which is computed during training. When tuning the model for regression tasks, choose this metric as the objective.

Metric Name	Description	Optimization Direction
<code>test:rmse</code>	Root Mean Square Error	Minimize

The Factorization Machines algorithm reports three binary classification metrics, which are computed during training. When tuning the model for binary classification tasks, choose one of these as the objective.

Metric Name	Description	Optimization Direction
<code>test:binary_classifier_accuracy</code>	Accuracy	Maximize
<code>test:binary_classifier_cross_entropy</code>	Cross entropy	Minimize
<code>test:binary_f_beta</code>	Beta	Maximize

### Tunable Factorization Machines Hyperparameters

You can tune the following hyperparameters for the Factorization Machines algorithm. The initialization parameters that contain the terms bias, linear, and factorization depend on their initialization method. There are three initialization methods: `uniform`, `normal`, and `constant`. These initialization methods are not themselves tunable. The parameters that are tunable are dependent on this choice of the initialization method. For example, if the initialization method is `uniform`, then only the `scale` parameters are tunable. Specifically, if `bias_init_method==uniform`, then `bias_init_scale`, `linear_init_scale`, and `factors_init_scale` are tunable. Similarly, if the initialization method is `normal`, then only `sigma` parameters are tunable. If the initialization method is `constant`, then only `value` parameters are tunable. These dependencies are listed in the following table.

Parameter Name	Parameter Type	Recommended Ranges	Dependency
<code>bias_init_scale</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	<code>bias_init_method==uniform</code>
<code>bias_init_sigma</code>	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	<code>bias_init_method==normal</code>

Parameter Name	Parameter Type	Recommended Ranges	Dependency
bias_init_value	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==constant
bias_lr	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None
bias_wd	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None
epoch	IntegerParameterRange	MinValue: 1, MaxValue: 1000	None
factors_init_scale	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==uniform
factors_init_sigma	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==normal
factors_init_value	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==constant
factors_lr	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None
factors_wd	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512]	None
linear_init_scale	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==uniform
linear_init_sigma	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==normal
linear_init_value	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	bias_init_method==constant
linear_lr	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None
linear_wd	ContinuousParameterRange	MinValue: 1e-8, MaxValue: 512	None
mini_batch_size	IntegerParameterRange	MinValue: 100, MaxValue: 10000	None

## Factorization Machines Response Formats

### JSON Response Format

Binary classification

```
let response = {
  "predictions": [
    {
      "score": 0.4,
      "predicted_label": 0
    }
  ]
}
```

```
    ]  
}
```

## Regression

```
let response = {  
    "predictions": [  
        {  
            "score": 0.4  
        }  
    ]  
}
```

## JSONLINES Response Format

### Binary classification

```
{"score": 0.4, "predicted_label": 0}
```

### Regression

```
{"score": 0.4}
```

## RECORDIO Response Format

### Binary classification

```
[  
    Record = {  
        features = {},  
        label = {  
            'score': {  
                keys: [],  
                values: [0.4] # float32  
            },  
            'predicted_label': {  
                keys: [],  
                values: [0.0] # float32  
            }  
        }  
    }  
]
```

### Regression

```
[  
    Record = {  
        features = {},  
        label = {  
            'score': {  
                keys: [],  
                values: [0.4] # float32  
            }  
        }  
    }  
]
```

## Image Classification Algorithm

The Amazon SageMaker image classification algorithm is a supervised learning algorithm that supports multi-label classification. It takes an image as input and outputs one or more labels assigned to that image. It uses a convolutional neural network (ResNet) that can be trained from scratch or trained using transfer learning when a large number of training images are not available.

The recommended input format for the Amazon SageMaker image classification algorithms is Apache MXNet [RecordIO](#). However, you can also use raw images in .jpg or .png format. Refer to [this discussion](#) for a broad overview of efficient data preparation and loading for machine learning systems.

**Note**

To maintain better interoperability with existing deep learning frameworks, this differs from the protobuf data formats commonly used by other Amazon SageMaker algorithms.

For more information on convolutional networks, see:

- [Deep residual learning for image recognition](#) Kaiming He, et al., 2016 IEEE Conference on Computer Vision and Pattern Recognition
- [ImageNet image database](#)
- [Image classification with Gluon-CV and MXNet](#)

**Topics**

- [Input/Output Interface for the Image Classification Algorithm \(p. 1399\)](#)
- [EC2 Instance Recommendation for the Image Classification Algorithm \(p. 1401\)](#)
- [Image Classification Sample Notebooks \(p. 1402\)](#)
- [How Image Classification Works \(p. 1402\)](#)
- [Image Classification Hyperparameters \(p. 1402\)](#)
- [Tune an Image Classification Model \(p. 1409\)](#)

## Input/Output Interface for the Image Classification Algorithm

The SageMaker Image Classification algorithm supports both RecordIO (`application/x-recordio`) and image (`image/png`, `image/jpeg`, and `application/x-image`) content types for training in file mode, and supports the RecordIO (`application/x-recordio`) content type for training in pipe mode. However, you can also train in pipe mode using the image files (`image/png`, `image/jpeg`, and `application/x-image`), without creating RecordIO files, by using the augmented manifest format.

Distributed training is supported for file mode and pipe mode. When using the RecordIO content type in pipe mode, you must set the `S3DataDistributionType` of the `S3DataSource` to `FullyReplicated`. The algorithm supports a fully replicated model where your data is copied onto each machine.

The algorithm supports `image/png`, `image/jpeg`, and `application/x-image` for inference.

### Train with RecordIO Format

If you use the RecordIO format for training, specify both `train` and `validation` channels as values for the `InputDataConfig` parameter of the [`CreateTrainingJob`](#) request. Specify one RecordIO (`.rec`) file in the `train` channel and one RecordIO file in the `validation` channel. Set the content type for both channels to `application/x-recordio`.

### Train with Image Format

If you use the Image format for training, specify `train`, `validation`, `train_lst`, and `validation_lst` channels as values for the `InputDataConfig` parameter of the [`CreateTrainingJob`](#) request. Specify the individual image data (`.jpg` or `.png` files) for the `train`

and validation channels. Specify one .lst file in each of the `train_lst` and `validation_lst` channels. Set the content type for all four channels to `application/x-image`.

**Note**

SageMaker reads the training and validation data separately from different channels, so you must store the training and validation data in different folders.

A .lst file is a tab-separated file with three columns that contains a list of image files. The first column specifies the image index, the second column specifies the class label index for the image, and the third column specifies the relative path of the image file. The image index in the first column must be unique across all of the images. The set of class label indices are numbered successively and the numbering should start with 0. For example, 0 for the cat class, 1 for the dog class, and so on for additional classes.

The following is an example of a .lst file:

```
5      1    your_image_directory/train_img_dog1.jpg
1000   0    your_image_directory/train_img_cat1.jpg
22     1    your_image_directory/train_img_dog2.jpg
```

For example, if your training images are stored in `s3://<your_bucket>/train/class_dog`, `s3://<your_bucket>/train/class_cat`, and so on, specify the path for your train channel as `s3://<your_bucket>/train`, which is the top-level directory for your data. In the .lst file, specify the relative path for an individual file named `train_image_dog1.jpg` in the `class_dog` class directory as `class_dog/train_image_dog1.jpg`. You can also store all your image files under one subdirectory inside the `train` directory. In that case, use that subdirectory for the relative path. For example, `s3://<your_bucket>/train/your_image_directory`.

### Train with Augmented Manifest Image Format

The augmented manifest format enables you to do training in Pipe mode using image files without needing to create RecordIO files. You need to specify both train and validation channels as values for the `InputDataConfig` parameter of the `CreateTrainingJob` request. While using the format, an S3 manifest file needs to be generated that contains the list of images and their corresponding annotations. The manifest file format should be in [JSON Lines](#) format in which each line represents one sample. The images are specified using the '`source-ref`' tag that points to the S3 location of the image. The annotations are provided under the "`AttributeNames`" parameter value as specified in the `CreateTrainingJob` request. It can also contain additional metadata under the `metadata` tag, but these are ignored by the algorithm. In the following example, the "`AttributeNames`" are contained in the list of image and annotation references `["source-ref", "class"]`. The corresponding label value is "0" for the first image and "1" for the second image:

```
{"source-ref": "s3://image/filename1.jpg", "class": "0"}
{"source-ref": "s3://image/filename2.jpg", "class": "1", "class-metadata": {"class-name": "cat", "type" : "groundtruth/image-classification"}}
```

The order of "`AttributeNames`" in the input files matters when training the `ImageClassification` algorithm. It accepts piped data in a specific order, with `image` first, followed by `label`. So the "`AttributeNames`" in this example are provided with "`source-ref`" first, followed by "`class`". When using the `ImageClassification` algorithm with Augmented Manifest, the value of the `RecordWrapperType` parameter must be "`RecordIO`".

Multi-label training is also supported by specifying a JSON array of values. The `num_classes` hyperparameter must be set to match the total number of classes. There are two valid label formats: multi-hot and class-id.

In the multi-hot format, each label is a multi-hot encoded vector of all classes, where each class takes the value of 0 or 1. In the following example, there are three classes. The first image is labeled with classes 0 and 2, while the second image is labeled with class 2 only:

```
{"image-ref": "s3://mybucket/sample01/image1.jpg", "class": "[1, 0, 1]"}  
 {"image-ref": "s3://mybucket/sample02/image2.jpg", "class": "[0, 0, 1]"}
```

In the class-id format, each label is a list of the class ids, from [0, num\_classes), which apply to the data point. The previous example would instead look like this:

```
{"image-ref": "s3://mybucket/sample01/image1.jpg", "class": "[0, 2]"}  
 {"image-ref": "s3://mybucket/sample02/image2.jpg", "class": "[2]"}
```

The multi-hot format is the default, but can be explicitly set in the content type with the `label-format` parameter: "application/x-recordio; label-format=multi-hot". The class-id format, which is the format outputted by `GroundTruth`, must be set explicitly: "application/x-recordio; label-format=class-id".

For more information on augmented manifest files, see [Provide Dataset Metadata to Training Jobs with an Augmented Manifest File \(p. 1864\)](#).

### Incremental Training

You can also seed the training of a new model with the artifacts from a model that you trained previously with SageMaker. Incremental training saves training time when you want to train a new model with the same or similar data. SageMaker image classification models can be seeded only with another built-in image classification model trained in SageMaker.

To use a pretrained model, in the `CreateTrainingJob` request, specify the `ChannelName` as "model" in the `InputDataConfig` parameter. Set the `ContentType` for the model channel to `application/x-sagemaker-model`. The input hyperparameters of both the new model and the pretrained model that you upload to the model channel must have the same settings for the `num_layers`, `image_shape` and `num_classes` input parameters. These parameters define the network architecture. For the pretrained model file, use the compressed model artifacts (in .tar.gz format) output by SageMaker. You can use either RecordIO or image formats for input data.

For a sample notebook that shows how to use incremental training with the SageMaker image classification algorithm, see the [End-to-End Incremental Training Image Classification Example](#). For more information on incremental training and for instructions on how to use it, see [Incremental Training in Amazon SageMaker \(p. 1855\)](#).

### Inference with the Image Classification Algorithm

The generated models can be hosted for inference and support encoded .jpg and .png image formats as `image/png`, `image/jpeg`, and `application/x-image` content-type. The input image is resized automatically. The output is the probability values for all classes encoded in JSON format, or in [JSON Lines text format](#) for batch transform. The image classification model processes a single image per request and so outputs only one line in the JSON or JSON Lines format. The following is an example of a response in JSON Lines format:

```
accept: application/jsonlines  
  
{"prediction": [prob_0, prob_1, prob_2, prob_3, ...]}
```

For more details on training and inference, see the image classification sample notebook instances referenced in the introduction.

### EC2 Instance Recommendation for the Image Classification Algorithm

For image classification, we support the following GPU instances for training: `m1.p2.xlarge`, `m1.p2.8xlarge`, `m1.p2.16xlarge`, `m1.p3.2xlarge`, `m1.p3.8xlarge` and `m1.p3.16xlarge`. We recommend using GPU instances with more memory for training with large batch sizes. However, both

CPU (such as C4) and GPU (such as P2 and P3) instances can be used for the inference. You can also run the algorithm on multi-GPU and multi-machine settings for distributed training.

Both P2 and P3 instances are supported in the image classification algorithm.

## Image Classification Sample Notebooks

For a sample notebook that uses the SageMaker image classification algorithm to train a model on the [caltech-256 dataset](#) and then to deploy it to perform inferences, see the [End-to-End Multiclass Image Classification Example](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the SageMaker samples. The example image classification notebooks are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

## How Image Classification Works

The image classification algorithm takes an image as input and classifies it into one of the output categories. Deep learning has revolutionized the image classification domain and has achieved great performance. Various deep learning networks such as ResNet [1], DenseNet, inception, and so on, have been developed to be highly accurate for image classification. At the same time, there have been efforts to collect labeled image data that are essential for training these networks. ImageNet[2] is one such large dataset that has more than 11 million images with about 11,000 categories. Once a network is trained with ImageNet data, it can then be used to generalize with other datasets as well, by simple re-adjustment or fine-tuning. In this transfer learning approach, a network is initialized with weights (in this example, trained on ImageNet), which can be later fine-tuned for an image classification task in a different dataset.

Image classification in Amazon SageMaker can be run in two modes: full training and transfer learning. In full training mode, the network is initialized with random weights and trained on user data from scratch. In transfer learning mode, the network is initialized with pre-trained weights and just the top fully connected layer is initialized with random weights. Then, the whole network is fine-tuned with new data. In this mode, training can be achieved even with a smaller dataset. This is because the network is already trained and therefore can be used in cases without sufficient training data.

## Image Classification Hyperparameters

Hyperparameters are parameters that are set before a machine learning model begins learning. The following hyperparameters are supported by the Amazon SageMaker built-in Image Classification algorithm. See [Tune an Image Classification Model \(p. 1409\)](#) for information on image classification hyperparameter tuning.

Parameter Name	Description
num_classes	<p>Number of output classes. This parameter defines the dimensions of the network output and is typically set to the number of classes in the dataset.</p> <p>Besides multi-class classification, multi-label classification is supported too. Please refer to <a href="#">Input/Output Interface for the Image Classification Algorithm (p. 1399)</a> for details on how to work with multi-label classification with augmented manifest files.</p> <p><b>Required</b></p> <p>Valid values: positive integer</p>
num_training_samples	Number of training examples in the input dataset.

Parameter Name	Description
	<p>If there is a mismatch between this value and the number of samples in the training set, then the behavior of the <code>lr_scheduler_step</code> parameter is undefined and distributed training accuracy might be affected.</p> <p><b>Required</b></p> <p>Valid values: positive integer</p>
<code>augmentation_type</code>	<p>Data augmentation type. The input images can be augmented in multiple ways as specified below.</p> <ul style="list-style-type: none"> <li>• <code>crop</code>: Randomly crop the image and flip the image horizontally</li> <li>• <code>crop_color</code>: In addition to '<code>crop</code>', three random values in the range [-36, 36], [-50, 50], and [-50, 50] are added to the corresponding Hue-Saturation-Lightness channels respectively</li> <li>• <code>crop_color_transform</code>: In addition to <code>crop_color</code>, random transformations, including rotation, shear, and aspect ratio variations are applied to the image. The maximum angle of rotation is 10 degrees, the maximum shear ratio is 0.1, and the maximum aspect changing ratio is 0.25.</li> </ul> <p><b>Optional</b></p> <p>Valid values: <code>crop</code>, <code>crop_color</code>, or <code>crop_color_transform</code>.</p> <p>Default value: no default value</p>
<code>beta_1</code>	<p>The beta1 for adam, that is the exponential decay rate for the first moment estimates.</p> <p><b>Optional</b></p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.9</p>
<code>beta_2</code>	<p>The beta2 for adam, that is the exponential decay rate for the second moment estimates.</p> <p><b>Optional</b></p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.999</p>

Parameter Name	Description
<code>checkpoint_frequency</code>	<p>Period to store model parameters (in number of epochs). Note that all checkpoint files are saved as part of the final model file "model.tar.gz" and uploaded to S3 to the specified model location. This increases the size of the model file proportionally to the number of checkpoints saved during training.</p> <p><b>Optional</b></p> <p>Valid values: positive integer no greater than epochs.</p> <p>Default value: no default value (Save checkpoint at the epoch that has the best validation accuracy)</p>
<code>early_stopping</code>	<p>True to use early stopping logic during training. False not to use it.</p> <p><b>Optional</b></p> <p>Valid values: True or False</p> <p>Default value: False</p>
<code>early_stopping_min_epochs</code>	<p>The minimum number of epochs that must be run before the early stopping logic can be invoked. It is used only when <code>early_stopping = True</code>.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 10</p>
<code>early_stopping_patience</code>	<p>The number of epochs to wait before ending training if no improvement is made in the relevant metric. It is used only when <code>early_stopping = True</code>.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 5</p>
<code>early_stopping_tolerance</code>	<p>Relative tolerance to measure an improvement in accuracy validation metric. If the ratio of the improvement in accuracy divided by the previous best accuracy is smaller than the <code>early_stopping_tolerance</code> value set, early stopping considers there is no improvement. It is used only when <code>early_stopping = True</code>.</p> <p><b>Optional</b></p> <p>Valid values: <math>0 \leq \text{float} \leq 1</math></p> <p>Default value: 0.0</p>

Parameter Name	Description
epochs	<p>Number of training epochs.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 30</p>
eps	<p>The epsilon for adam and rmsprop. It is usually set to a small value to avoid division by 0.</p> <p><b>Optional</b></p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 1e-8</p>
gamma	<p>The gamma for rmsprop, the decay factor for the moving average of the squared gradient.</p> <p><b>Optional</b></p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.9</p>
image_shape	<p>The input image dimensions, which is the same size as the input layer of the network. The format is defined as 'num_channels, height, width'. The image dimension can take on any value as the network can handle varied dimensions of the input. However, there may be memory constraints if a larger image dimension is used. Pretrained models can use only a fixed 224 x 224 image size. Typical image dimensions for image classification are '3, 224, 224'. This is similar to the ImageNet dataset.</p> <p>For training, if any input image is smaller than this parameter in any dimension, training fails. If an image is larger, a portion of the image is cropped, with the cropped area specified by this parameter. If hyperparameter augmentation_type is set, random crop is taken; otherwise, central crop is taken.</p> <p>At inference, input images are resized to the image_shape that was used during training. Aspect ratio is not preserved, and images are not cropped.</p> <p><b>Optional</b></p> <p>Valid values: string</p> <p>Default value: '3, 224, 224'</p>

Parameter Name	Description
<code>kv_store</code>	<p>Weight update synchronization mode during distributed training. The weight updates can be updated either synchronously or asynchronously across machines. Synchronous updates typically provide better accuracy than asynchronous updates but can be slower. See <a href="#">distributed training in MXNet</a> for more details.</p> <p>This parameter is not applicable to single machine training.</p> <ul style="list-style-type: none"> <li>• <code>dist_sync</code>: The gradients are synchronized after every batch with all the workers. With <code>dist_sync</code>, batch-size now means the batch size used on each machine. So if there are n machines and we use batch size b, then <code>dist_sync</code> behaves like local with batch size <math>n \times b</math></li> <li>• <code>dist_async</code>: Performs asynchronous updates. The weights are updated whenever gradients are received from any machine and the weight updates are atomic. However, the order is not guaranteed.</li> </ul> <p><b>Optional</b></p> <p>Valid values: <code>dist_sync</code> or <code>dist_async</code></p> <p>Default value: no default value</p>
<code>learning_rate</code>	<p>Initial learning rate.</p> <p><b>Optional</b></p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.1</p>
<code>lr_scheduler_factor</code>	<p>The ratio to reduce learning rate used in conjunction with the <code>lr_scheduler_step</code> parameter, defined as <math>lr_{new} = lr_{old} * lr\_scheduler\_factor</math>.</p> <p><b>Optional</b></p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.1</p>
<code>lr_scheduler_step</code>	<p>The epochs at which to reduce the learning rate. As explained in the <code>lr_scheduler_factor</code> parameter, the learning rate is reduced by <code>lr_scheduler_factor</code> at these epochs. For example, if the value is set to "10, 20", then the learning rate is reduced by <code>lr_scheduler_factor</code> after 10th epoch and again by <code>lr_scheduler_factor</code> after 20th epoch. The epochs are delimited by ",".</p> <p><b>Optional</b></p> <p>Valid values: string</p> <p>Default value: no default value</p>

Parameter Name	Description
<code>mini_batch_size</code>	<p>The batch size for training. In a single-machine multi-GPU setting, each GPU handles <code>mini_batch_size/num_gpu</code> training samples. For the multi-machine training in <code>dist_sync</code> mode, the actual batch size is <code>mini_batch_size*number of machines</code>. See MXNet docs for more details.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 32</p>
<code>momentum</code>	<p>The momentum for <code>sgd</code> and <code>nag</code>, ignored for other optimizers.</p> <p><b>Optional</b></p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.9</p>
<code>multi_label</code>	<p>Flag to use for multi-label classification where each sample can be assigned multiple labels. Average accuracy across all classes is logged.</p> <p><b>Optional</b></p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>
<code>num_layers</code>	<p>Number of layers for the network. For data with large image size (for example, 224x224 - like ImageNet), we suggest selecting the number of layers from the set [18, 34, 50, 101, 152, 200]. For data with small image size (for example, 28x28 - like CIFAR), we suggest selecting the number of layers from the set [20, 32, 44, 56, 110]. The number of layers in each set is based on the ResNet paper. For transfer learning, the number of layers defines the architecture of base network and hence can only be selected from the set [18, 34, 50, 101, 152, 200].</p> <p><b>Optional</b></p> <p>Valid values: positive integer in [18, 34, 50, 101, 152, 200] or [20, 32, 44, 56, 110]</p> <p>Default value: 152</p>

Parameter Name	Description
optimizer	<p>The optimizer type. For more details of the parameters for the optimizers, please refer to MXNet's API.</p> <p><b>Optional</b></p> <p>Valid values: One of <code>sgd</code>, <code>adam</code>, <code>rmsprop</code>, or <code>nag</code>.</p> <ul style="list-style-type: none"> <li>• <code>sgd</code>: <a href="#">Stochastic gradient descent</a></li> <li>• <code>adam</code>: <a href="#">Adaptive momentum estimation</a></li> <li>• <code>rmsprop</code>: <a href="#">Root mean square propagation</a></li> <li>• <code>nag</code>: <a href="#">Nesterov accelerated gradient</a></li> </ul> <p>Default value: <code>sgd</code></p>
precision_dtype	<p>The precision of the weights used for training. The algorithm can use either single precision (<code>float32</code>) or half precision (<code>float16</code>) for the weights. Using half-precision for weights results in reduced memory consumption.</p> <p><b>Optional</b></p> <p>Valid values: <code>float32</code> or <code>float16</code></p> <p>Default value: <code>float32</code></p>
resize	<p>The number of pixels in the shortest side of an image after resizing it for training. If the parameter is not set, then the training data is used without resizing. The parameter should be larger than both the width and height components of <code>image_shape</code> to prevent training failure.</p> <p><b>Required</b> when using image content types</p> <p><b>Optional</b> when using the RecordIO content type</p> <p>Valid values: positive integer</p> <p>Default value: no default value</p>
top_k	<p>Reports the top-k accuracy during training. This parameter has to be greater than 1, since the top-1 training accuracy is the same as the regular training accuracy that has already been reported.</p> <p><b>Optional</b></p> <p>Valid values: positive integer larger than 1.</p> <p>Default value: no default value</p>

Parameter Name	Description
<code>use_pretrained_model</code>	<p>Flag to use pre-trained model for training. If set to 1, then the pretrained model with the corresponding number of layers is loaded and used for training. Only the top FC layer are reinitialized with random weights. Otherwise, the network is trained from scratch.</p> <p><b>Optional</b></p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>
<code>use_weighted_loss</code>	<p>Flag to use weighted cross-entropy loss for multi-label classification (used only when <code>multi_label</code> = 1), where the weights are calculated based on the distribution of classes.</p> <p><b>Optional</b></p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>
<code>weight_decay</code>	<p>The coefficient weight decay for <code>sgd</code> and <code>nag</code>, ignored for other optimizers.</p> <p><b>Optional</b></p> <p>Valid values: float. Range in [0, 1].</p> <p>Default value: 0.0001</p>

## Tune an Image Classification Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics Computed by the Image Classification Algorithm

The image classification algorithm is a supervised algorithm. It reports an accuracy metric that is computed during training. When tuning the model, choose this metric as the objective metric.

Metric Name	Description	Optimization Direction
<code>validation:accuracy</code>	The ratio of the number of correct predictions to the total number of predictions made.	Maximize

### Tunable Image Classification Hyperparameters

Tune an image classification model with the following hyperparameters. The hyperparameters that have the greatest impact on image classification objective metrics are: `mini_batch_size`, `learning_rate`,

and `optimizer`. Tune the optimizer-related hyperparameters, such as `momentum`, `weight_decay`, `beta_1`, `beta_2`, `eps`, and `gamma`, based on the selected `optimizer`. For example, use `beta_1` and `beta_2` only when `adam` is the optimizer.

For more information about which hyperparameters are used in each optimizer, see [Image Classification Hyperparameters \(p. 1402\)](#).

Parameter Name	Parameter Type	Recommended Ranges
<code>beta_1</code>	ContinuousParameterRanges	MinValue: 1e-6, MaxValue: 0.999
<code>beta_2</code>	ContinuousParameterRanges	MinValue: 1e-6, MaxValue: 0.999
<code>eps</code>	ContinuousParameterRanges	MinValue: 1e-8, MaxValue: 1.0
<code>gamma</code>	ContinuousParameterRanges	MinValue: 1e-8, MaxValue: 0.999
<code>learning_rate</code>	ContinuousParameterRanges	MinValue: 1e-6, MaxValue: 0.5
<code>mini_batch_size</code>	IntegerParameterRanges	MinValue: 8, MaxValue: 512
<code>momentum</code>	ContinuousParameterRanges	MinValue: 0.0, MaxValue: 0.999
<code>optimizer</code>	CategoricalParameterRanges	['sgd', 'adam', 'rmsprop', 'nag']
<code>weight_decay</code>	ContinuousParameterRanges	MinValue: 0.0, MaxValue: 0.999

## IP Insights

Amazon SageMaker IP Insights is an unsupervised learning algorithm that learns the usage patterns for IPv4 addresses. It is designed to capture associations between IPv4 addresses and various entities, such as user IDs or account numbers. You can use it to identify a user attempting to log into a web service from an anomalous IP address, for example. Or you can use it to identify an account that is attempting to create computing resources from an unusual IP address. Trained IP Insight models can be hosted at an endpoint for making real-time predictions or used for processing batch transforms.

SageMaker IP insights ingests historical data as (entity, IPv4 Address) pairs and learns the IP usage patterns of each entity. When queried with an (entity, IPv4 Address) event, a SageMaker IP Insights model returns a score that infers how anomalous the pattern of the event is. For example, when a user attempts to log in from an IP address, if the IP Insights score is high enough, a web login server might decide to trigger a multi-factor authentication system. In more advanced solutions, you can feed the IP Insights score into another machine learning model. For example, you can combine the IP Insight score with other features to rank the findings of another security system, such as those from [Amazon GuardDuty](#).

The SageMaker IP Insights algorithm can also learn vector representations of IP addresses, known as *embeddings*. You can use vector-encoded embeddings as features in downstream machine learning tasks that use the information observed in the IP addresses. For example, you can use them in tasks such as measuring similarities between IP addresses in clustering and visualization tasks.

## Topics

- [Input/Output Interface for the IP Insights Algorithm \(p. 1411\)](#)
- [EC2 Instance Recommendation for the IP Insights Algorithm \(p. 1411\)](#)
- [IP Insights Sample Notebooks \(p. 1412\)](#)
- [How IP Insights Works \(p. 1412\)](#)
- [IP Insights Hyperparameters \(p. 1413\)](#)
- [Tune an IP Insights Model \(p. 1416\)](#)
- [IP Insights Data Formats \(p. 1417\)](#)

## [Input/Output Interface for the IP Insights Algorithm](#)

### Training and Validation

The SageMaker IP Insights algorithm supports training and validation data channels. It uses the optional validation channel to compute an area-under-curve (AUC) score on a predefined negative sampling strategy. The AUC metric validates how well the model discriminates between positive and negative samples. Training and validation data content types need to be in `text/csv` format. The first column of the CSV data is an opaque string that provides a unique identifier for the entity. The second column is an IPv4 address in decimal-dot notation. IP Insights currently supports only File mode. For more information and some examples, see [IP Insights Training Data Formats \(p. 1417\)](#).

### Inference

For inference, IP Insights supports `text/csv`, `application/json`, and `application/jsonlines` data content types. For more information about the common data formats for inference provided by SageMaker, see [Common Data Formats for Inference \(p. 1358\)](#). IP Insights inference returns output formatted as either `application/json` or `application/jsonlines`. Each record in the output data contains the corresponding `dot_product` (or compatibility score) for each input data point. For more information and some examples, see [IP Insights Inference Data Formats \(p. 1418\)](#).

## [EC2 Instance Recommendation for the IP Insights Algorithm](#)

The SageMaker IP Insights algorithm can run on both GPU and CPU instances. For training jobs, we recommend using GPU instances. However, for certain workloads with large training datasets, distributed CPU instances might reduce training costs. For inference, we recommend using CPU instances.

### [GPU Instances for the IP Insights Algorithm](#)

IP Insights supports all available GPUs. If you need to speed up training, we recommend starting with a single GPU instance, such as `ml.p3.2xlarge`, and then moving to a multi-GPU environment, such as `ml.p3.8xlarge` and `ml.p3.16xlarge`. Multi-GPUs automatically divide the mini batches of training data across themselves. If you switch from a single GPU to multiple GPUs, the `mini_batch_size` is divided equally into the number of GPUs used. You may want to increase the value of the `mini_batch_size` to compensate for this.

### [CPU Instances for the IP Insights Algorithm](#)

The type of CPU instance that we recommend depends largely on the instance's available memory and the model size. The model size is determined by two hyperparameters: `vector_dim` and `num_entity_vectors`. The maximum supported model size is 8 GB. The following table lists typical EC2 instance types that you would deploy based on these input parameters for various model sizes. In Table 1, the value for `vector_dim` in the first column range from 32 to 2048 and the values for `num_entity_vectors` in the first row range from 10,000 to 50,000,000.

<code>vector_d</code>	10,000	50,000	100,000	500,000	1,000,000	5,000,000	10,000,000	50,000,000
\								
<code>num_entities</code>								
32	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.xlarge	ml.m5.2xlarge	ml.m5.4xlarge
64	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.2xlarge	ml.m5.4xlarge	
128	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.2xlarge	ml.m5.4xlarge		
256	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.xlarge	ml.m5.4xlarge		
512	ml.m5.large	ml.m5.large	ml.m5.large	ml.m5.2xlarge				
1024	ml.m5.large	ml.m5.large	ml.m5.xlarge	ml.m5.4xlarge				
2048	ml.m5.large	ml.m5.xlarge	ml.m5.xlarge					

The values for the `mini_batch_size`, `num_ip_encoder_layers`, `random_negative_sampling_rate`, and `shuffled_negative_sampling_rate` hyperparameters also affect the amount of memory required. If these values are large, you might need to use a larger instance type than normal.

## IP Insights Sample Notebooks

For a sample notebook that shows how to train the SageMaker IP Insights algorithm and perform inferences with it, see [An Introduction to the SageMaker IP Insights Algorithm](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). After creating a notebook instance, choose the **SageMaker Examples** tab to see a list of all the SageMaker examples. To open a notebook, choose its **Use** tab and choose **Create copy**.

## How IP Insights Works

Amazon SageMaker IP Insights is an unsupervised algorithm that consumes observed data in the form of (entity, IPv4 address) pairs that associates entities with IP addresses. IP Insights determines how likely it is that an entity would use a particular IP address by learning latent vector representations for both entities and IP addresses. The distance between these two representations can then serve as the proxy for how likely this association is.

The IP Insights algorithm uses a neural network to learn the latent vector representations for entities and IP addresses. Entities are first hashed to a large but fixed hash space and then encoded by a simple embedding layer. Character strings such as user names or account IDs can be fed directly into IP Insights as they appear in log files. You don't need to preprocess the data for entity identifiers. You can provide entities as an arbitrary string value during both training and inference. The hash size should be configured with a value that is high enough to insure that the number of *collisions*, which occur when distinct entities are mapped to the same latent vector, remain insignificant. For more information about how to select appropriate hash sizes, see [Feature Hashing for Large Scale Multitask Learning](#). For representing IP addresses, on the other hand, IP Insights uses a specially designed encoder network to uniquely represent each possible IPv4 address by exploiting the prefix structure of IP addresses.

During training, IP Insights automatically generates negative samples by randomly pairing entities and IP addresses. These negative samples represent data that is less likely to occur in reality. The model is trained to discriminate between positive samples that are observed in the training data and these generated negative samples. More specifically, the model is trained to minimize the *cross entropy*, also known as the *log loss*, defined as follows:

$$L = \frac{1}{N} \sum_n [y_n \log p_n + (1 - y_n) \log (1 - p_n)]$$

$y_n$  is the label that indicates whether the sample is from the real distribution governing observed data ( $y_n=1$ ) or from the distribution generating negative samples ( $y_n=0$ ).  $p_n$  is the probability that the sample is from the real distribution, as predicted by the model.

Generating negative samples is an important process that is used to achieve an accurate model of the observed data. If negative samples are extremely unlikely, for example, if all of the IP addresses in negative samples are 10.0.0.0, then the model trivially learns to distinguish negative samples and fails to accurately characterize the actual observed dataset. To keep negative samples more realistic, IP Insights generates negative samples both by randomly generating IP addresses and randomly picking IP addresses from training data. You can configure the type of negative sampling and the rates at which negative samples are generated with the `random_negative_sampling_rate` and `shuffled_negative_sampling_rate` hyperparameters.

Given an nth (entity, IP address pair), the IP Insights model outputs a *score*,  $S_n$ , that indicates how compatible the entity is with the IP address. This score corresponds to the log odds ratio for a given (entity, IP address) of the pair coming from a real distribution as compared to coming from a negative distribution. It is defined as follows:

$$S_n = \log \left( \frac{P_{\text{real}}(n)}{P_{\text{neg}}(n)} \right)$$

The score is essentially a measure of the similarity between the vector representations of the nth entity and IP address. It can be interpreted as how much more likely it would be to observe this event in reality than in a randomly generated dataset. During training, the algorithm uses this score to calculate an estimate of the probability of a sample coming from the real distribution,  $p_n$ , to use in the cross entropy minimization, where:

$$p_n = \frac{1}{1 + e^{-S_n}}$$

## IP Insights Hyperparameters

In the [CreateTransformJob](#) request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the Amazon SageMaker IP Insights algorithm.

Parameter Name	Description
<code>num_entity_vectors</code>	<p>The number of entity vector representations (entity embedding vectors) to train. Each entity in the training set is randomly assigned to one of these vectors using a hash function. Because of hash collisions, it might be possible to have multiple entities assigned to the same vector. This would cause the same vector to represent multiple entities. This generally has a negligible effect on model performance, as long as the collision rate is not too severe. To keep the collision rate low, set this value as high as possible. However, the model size, and, therefore, the memory requirement, for both training and inference, scales linearly with this hyperparameter. We recommend that you set this value to twice the number of unique entity identifiers.</p> <p><b>Required</b></p> <p>Valid values: 1 ≤ positive integer ≤ 250,000,000</p>

Parameter Name	Description
vector_dim	<p>The size of embedding vectors to represent entities and IP addresses. The larger the value, the more information that can be encoded using these representations. In practice, model size scales linearly with this parameter and limits how large the dimension can be. In addition, using vector representations that are too large can cause the model to overfit, especially for small training datasets. Overfitting occurs when a model doesn't learn any pattern in the data but effectively memorizes the training data and, therefore, cannot generalize well and performs poorly during inference. The recommended value is 128.</p> <p><b>Required</b></p> <p>Valid values: 4 ≤ positive integer ≤ 4096</p>
batch_metrics_publish_interval	<p>The interval (every X batches) at which the Apache MXNet Speedometer function prints the training speed of the network (samples/second).</p> <p><b>Optional</b></p> <p>Valid values: positive integer ≥ 1</p> <p>Default value: 1,000</p>
epochs	<p>The number of passes over the training data. The optimal value depends on your data size and learning rate. Typical values range from 5 to 100.</p> <p><b>Optional</b></p> <p>Valid values: positive integer ≥ 1</p> <p>Default value: 10</p>
learning_rate	<p>The learning rate for the optimizer. IP Insights use a gradient-descent-based Adam optimizer. The learning rate effectively controls the step size to update model parameters at each iteration. Too large a learning rate can cause the model to diverge because the training is likely to overshoot a minima. On the other hand, too small a learning rate slows down convergence. Typical values range from 1e-4 to 1e-1.</p> <p><b>Optional</b></p> <p>Valid values: 1e-6 ≤ float ≤ 10.0</p> <p>Default value: 0.001</p>

Parameter Name	Description
<code>mini_batch_size</code>	<p>The number of examples in each mini batch. The training procedure processes data in mini batches. The optimal value depends on the number of unique account identifiers in the dataset. In general, the larger the <code>mini_batch_size</code>, the faster the training and the greater the number of possible shuffled-negative-sample combinations. However, with a large <code>mini_batch_size</code>, the training is more likely to converge to a poor local minimum and perform relatively worse for inference.</p> <p><b>Optional</b></p> <p>Valid values: <math>1 \leq \text{positive integer} \leq 500000</math></p> <p>Default value: 10,000</p>
<code>num_ip_encoder_layers</code>	<p>The number of fully connected layers used to encode the IP address embedding. The larger the number of layers, the greater the model's capacity to capture patterns among IP addresses. However, using a large number of layers increases the chance of overfitting.</p> <p><b>Optional</b></p> <p>Valid values: <math>0 \leq \text{positive integer} \leq 100</math></p> <p>Default value: 1</p>
<code>random_negative_sampling_rate</code>	<p>The number of random negative samples, R, to generate per input example. The training procedure relies on negative samples to prevent the vector representations of the model collapsing to a single point. Random negative sampling generates R random IP addresses for each input account in the mini batch. The sum of the <code>random_negative_sampling_rate</code> (R) and <code>shuffled_negative_sampling_rate</code> (S) must be in the interval: <math>1 \leq R + S \leq 500</math>.</p> <p><b>Optional</b></p> <p>Valid values: <math>0 \leq \text{positive integer} \leq 500</math></p> <p>Default value: 1</p>

Parameter Name	Description
shuffled_negative_sampling_rate	<p>The number of shuffled negative samples, S, to generate per input example. In some cases, it helps to use more realistic negative samples that are randomly picked from the training data itself. This kind of negative sampling is achieved by shuffling the data within a mini batch. Shuffled negative sampling generates S negative IP addresses by shuffling the IP address and account pairings within a mini batch. The sum of the <code>random_negative_sampling_rate</code> (R) and <code>shuffled_negative_sampling_rate</code> (S) must be in the interval: <math>1 \leq R + S \leq 500</math>.</p> <p><b>Optional</b></p> <p>Valid values: <math>0 \leq \text{positive integer} \leq 500</math></p> <p>Default value: 1</p>
weight_decay	<p>The weight decay coefficient. This parameter adds an L2 regularization factor that is required to prevent the model from overfitting the training data.</p> <p><b>Optional</b></p> <p>Valid values: <math>0.0 \leq \text{float} \leq 10.0</math></p> <p>Default value: 0.00001</p>

## Tune an IP Insights Model

*Automatic model tuning*, also called hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics Computed by the IP Insights Algorithm

The Amazon SageMaker IP Insights algorithm is an unsupervised learning algorithm that learns associations between IP addresses and entities. The algorithm trains a discriminator model, which learns to separate observed data points (*positive samples*) from randomly generated data points (*negative samples*). Automatic model tuning on IP Insights helps you find the model that can most accurately distinguish between unlabeled validation data and automatically generated negative samples. The model accuracy on the validation dataset is measured by the area under the receiver operating characteristic curve. This `validation:discriminator_auc` metric can take values between 0.0 and 1.0, where 1.0 indicates perfect accuracy.

The IP Insights algorithm computes a `validation:discriminator_auc` metric during validation, the value of which is used as the objective function to optimize for hyperparameter tuning.

Metric Name	Description	Optimization Direction
validation:discriminability	Area under the receiver operating characteristic curve on the validation dataset. The validation dataset is not labeled. Area Under the Curve (AUC) is a metric that describes the model's ability to discriminate validation data points from randomly generated data points.	Maximize

## Tunable IP Insights Hyperparameters

You can tune the following hyperparameters for the SageMaker IP Insights algorithm.

Parameter Name	Parameter Type	Recommended Ranges
epochs	IntegerParameterRange	MinValue: 1, MaxValue: 100
learning_rate	ContinuousParameterRange	MinValue: 1e-4, MaxValue: 0.1
mini_batch_size	IntegerParameterRanges	MinValue: 100, MaxValue: 50000
num_entity_vectors	IntegerParameterRanges	MinValue: 10000, MaxValue: 1000000
num_ip_encoder_layers	IntegerParameterRanges	MinValue: 1, MaxValue: 10
random_negative_samples	IntegerParameterRanges	MinValue: 0, MaxValue: 10
shuffled_negative_samples	IntegerParameterRanges	MinValue: 0, MaxValue: 10
vector_dim	IntegerParameterRanges	MinValue: 8, MaxValue: 256
weight_decay	ContinuousParameterRange	MinValue: 0.0, MaxValue: 1.0

## IP Insights Data Formats

This section provides examples of the available input and output data formats used by the IP Insights algorithm during training and inference.

### Topics

- [IP Insights Training Data Formats \(p. 1417\)](#)
- [IP Insights Inference Data Formats \(p. 1418\)](#)

### IP Insights Training Data Formats

The following are the available data input formats for the IP Insights algorithm. Amazon SageMaker built-in algorithms adhere to the common input training format described in [Common Data Formats for](#)

[Training \(p. 1354\)](#). However, the SageMaker IP Insights algorithm currently supports only the CSV data input format.

## IP Insights Training Data Input Formats

### INPUT: CSV

The CSV file must have two columns. The first column is an opaque string that corresponds to an entity's unique identifier. The second column is the IPv4 address of the entity's access event in decimal-dot notation.

content-type: text/csv

```
entity_id_1, 192.168.1.2
entity_id_2, 10.10.1.2
```

## IP Insights Inference Data Formats

The following are the available input and output formats for the IP Insights algorithm. Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats for Inference \(p. 1358\)](#). However, the SageMaker IP Insights algorithm does not currently support RecordIO format.

### IP Insights Input Request Formats

#### INPUT: CSV Format

The CSV file must have two columns. The first column is an opaque string that corresponds to an entity's unique identifier. The second column is the IPv4 address of the entity's access event in decimal-dot notation.

content-type: text/csv

```
entity_id_1, 192.168.1.2
entity_id_2, 10.10.1.2
```

#### INPUT: JSON Format

JSON data can be provided in different formats. IP Insights follows the common SageMaker formats. For more information about inference formats, see [Common Data Formats for Inference \(p. 1358\)](#).

content-type: application/json

```
{
  "instances": [
    {"data": {"features": {"values": ["entity_id_1", "192.168.1.2"]}}},
    {"features": ["entity_id_2", "10.10.1.2"]}
  ]
}
```

#### INPUT: JSONLINES Format

The JSON Lines content type is useful for running batch transform jobs. For more information on SageMaker inference formats, see [Common Data Formats for Inference \(p. 1358\)](#). For more information on running batch transform jobs, see [Get Inferences for an Entire Dataset with Batch Transform \(p. 13\)](#).

content-type: application/jsonlines

```
{"data": {"features": {"values": ["entity_id_1", "192.168.1.2"]}}},  
{"features": ["entity_id_2", "10.10.1.2"]}]
```

## IP Insights Output Response Formats

### OUTPUT: JSON Response Format

The default output of the SageMaker IP Insights algorithm is the dot\_product between the input entity and IP address. The dot\_product signifies how compatible the model considers the entity and IP address. The dot\_product is unbounded. To make predictions about whether an event is anomalous, you need to set a threshold based on your defined distribution. For information about how to use the dot\_product for anomaly detection, see the [An Introduction to the SageMakerIP Insights Algorithm](#).

accept: application/json

```
{  
    "predictions": [  
        {"dot_product": 0.0},  
        {"dot_product": 2.0}  
    ]  
}
```

Advanced users can access the model's learned entity and IP embeddings by providing the additional content-type parameter verbose=True to the Accept heading. You can use the entity\_embedding and ip\_embedding for debugging, visualizing, and understanding the model. Additionally, you can use these embeddings in other machine learning techniques, such as classification or clustering.

accept: application/json;verbose=True

```
{  
    "predictions": [  
        {  
            "dot_product": 0.0,  
            "entity_embedding": [1.0, 0.0, 0.0],  
            "ip_embedding": [0.0, 1.0, 0.0]  
        },  
        {  
            "dot_product": 2.0,  
            "entity_embedding": [1.0, 0.0, 1.0],  
            "ip_embedding": [1.0, 0.0, 1.0]  
        }  
    ]  
}
```

### OUTPUT: JSONLINES Response Format

accept: application/jsonlines

```
{"dot_product": 0.0}  
{"dot_product": 2.0}
```

accept: application/jsonlines; verbose=True

```
{"dot_product": 0.0, "entity_embedding": [1.0, 0.0, 0.0], "ip_embedding": [0.0, 1.0, 0.0]}  
{"dot_product": 2.0, "entity_embedding": [1.0, 0.0, 1.0], "ip_embedding": [1.0, 0.0, 1.0]}
```

## K-Means Algorithm

K-means is an unsupervised learning algorithm. It attempts to find discrete groupings within data, where members of a group are as similar as possible to one another and as different as possible from members of other groups. You define the attributes that you want the algorithm to use to determine similarity.

Amazon SageMaker uses a modified version of the web-scale k-means clustering algorithm. Compared with the original version of the algorithm, the version used by Amazon SageMaker is more accurate. Like the original algorithm, it scales to massive datasets and delivers improvements in training time. To do this, the version used by Amazon SageMaker streams mini-batches (small, random subsets) of the training data. For more information about mini-batch k-means, see [Web-scale k-means Clustering](#).

The k-means algorithm expects tabular data, where rows represent the observations that you want to cluster, and the columns represent attributes of the observations. The  $n$  attributes in each row represent a point in  $n$ -dimensional space. The Euclidean distance between these points represents the similarity of the corresponding observations. The algorithm groups observations with similar attribute values (the points corresponding to these observations are closer together). For more information about how k-means works in Amazon SageMaker, see [How K-Means Clustering Works \(p. 1421\)](#).

### Topics

- [Input/Output Interface for the K-Means Algorithm \(p. 1420\)](#)
- [EC2 Instance Recommendation for the K-Means Algorithm \(p. 1420\)](#)
- [K-Means Sample Notebooks \(p. 1420\)](#)
- [How K-Means Clustering Works \(p. 1421\)](#)
- [K-Means Hyperparameters \(p. 1423\)](#)
- [Tune a K-Means Model \(p. 1426\)](#)
- [K-Means Response Formats \(p. 1427\)](#)

## Input/Output Interface for the K-Means Algorithm

For training, the k-means algorithm expects data to be provided in the `train` channel (recommended `S3DataDistributionType=ShardedByS3Key`), with an optional `test` channel (recommended `S3DataDistributionType=FullyReplicated`) to score the data on. Both `recordIO-wrapped-protobuf` and `CSV` formats are supported for training. You can use either File mode or Pipe mode to train models on data that is formatted as `recordIO-wrapped-protobuf` or as `CSV`.

For inference, `text/csv`, `application/json`, and `application/x-recordio-protobuf` are supported. k-means returns a `closest_cluster` label and the `distance_to_cluster` for each observation.

For more information on input and output file formats, see [K-Means Response Formats \(p. 1427\)](#) for inference and the [K-Means Sample Notebooks \(p. 1420\)](#). The k-means algorithm does not support multiple instance learning, in which the training set consists of labeled “bags”, each of which is a collection of unlabeled instances.

## EC2 Instance Recommendation for the K-Means Algorithm

We recommend training k-means on CPU instances. You can train on GPU instances, but should limit GPU training to `p* .xlarge` instances because only one GPU per instance is used.

## K-Means Sample Notebooks

For a sample notebook that uses the SageMaker K-means algorithm to segment the population of counties in the United States by attributes identified using principle component analysis, see [Analyze US census data for population segmentation using Amazon SageMaker](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon](#)

[SageMaker Notebook Instances \(p. 120\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the SageMaker samples. To open a notebook, click on its **Use** tab and select **Create copy**.

## How K-Means Clustering Works

K-means is an algorithm that trains a model that groups similar objects together. The k-means algorithm accomplishes this by mapping each observation in the input dataset to a point in the  $n$ -dimensional space (where  $n$  is the number of attributes of the observation). For example, your dataset might contain observations of temperature and humidity in a particular location, which are mapped to points  $(t, h)$  in 2-dimensional space.

### Note

Clustering algorithms are unsupervised. In unsupervised learning, labels that might be associated with the objects in the training dataset aren't used.

In k-means clustering, each cluster has a center. During model training, the k-means algorithm uses the distance of the point that corresponds to each observation in the dataset to the cluster centers as the basis for clustering. You choose the number of clusters ( $k$ ) to create.

For example, suppose that you want to create a model to recognize handwritten digits and you choose the MNIST dataset for training. The dataset provides thousands of images of handwritten digits (0 through 9). In this example, you might choose to create 10 clusters, one for each digit (0, 1, ..., 9). As part of model training, the k-means algorithm groups the input images into 10 clusters.

Each image in the MNIST dataset is a 28x28-pixel image, with a total of 784 pixels. Each image corresponds to a point in a 784-dimensional space, similar to a point in a 2-dimensional space  $(x,y)$ . To find a cluster to which a point belongs, the k-means algorithm finds the distance of that point from all of the cluster centers. It then chooses the cluster with the closest center as the cluster to which the image belongs.

### Note

Amazon SageMaker uses a customized version of the algorithm where, instead of specifying that the algorithm create  $k$  clusters, you might choose to improve model accuracy by specifying extra cluster centers ( $K = k*x$ ). However, the algorithm ultimately reduces these to  $k$  clusters.

In SageMaker, you specify the number of clusters when creating a training job. For more information, see [CreateTrainingJob](#). In the request body, you add the `HyperParameters` string map to specify the `k` and `extra_center_factor` strings.

The following is a summary of how k-means works for model training in SageMaker:

1. It determines the initial  $K$  cluster centers.

### Note

In the following topics,  $K$  clusters refer to  $k * x$ , where you specify  $k$  and  $x$  when creating a model training job.

2. It iterates over input training data and recalculates cluster centers.
3. It reduces resulting clusters to  $k$  (if the data scientist specified the creation of  $k*x$  clusters in the request).

The following sections also explain some of the parameters that a data scientist might specify to configure a model training job as part of the `HyperParameters` string map.

### Topics

- [Step 1: Determine the Initial Cluster Centers \(p. 1422\)](#)
- [Step 2: Iterate over the Training Dataset and Calculate Cluster Centers \(p. 1422\)](#)
- [Step 3: Reduce the Clusters from  \$K\$  to  \$k\$  \(p. 1423\)](#)

## Step 1: Determine the Initial Cluster Centers

When using k-means in SageMaker, the initial cluster centers are chosen from the observations in a small, randomly sampled batch. Choose one of the following strategies to determine how these initial cluster centers are selected:

- The random approach—Randomly choose  $K$  observations in your input dataset as cluster centers. For example, you might choose a cluster center that points to the 784-dimensional space that corresponds to any 10 images in the MNIST training dataset.
- The k-means++ approach, which works as follows:
  1. Start with one cluster and determine its center. You randomly select an observation from your training dataset and use the point corresponding to the observation as the cluster center. For example, in the MNIST dataset, randomly choose a handwritten digit image. Then choose the point in the 784-dimensional space that corresponds to the image as your cluster center. This is cluster center 1.
  2. Determine the center for cluster 2. From the remaining observations in the training dataset, pick an observation at random. Choose one that is different than the one you previously selected. This observation corresponds to a point that is far away from cluster center 1. Using the MNIST dataset as an example, you do the following:
    - For each of the remaining images, find the distance of the corresponding point from cluster center 1. Square the distance and assign a probability that is proportional to the square of the distance. That way, an image that is different from the one that you previously selected has a higher probability of getting selected as cluster center 2.
    - Choose one of the images randomly, based on probabilities assigned in the previous step. The point that corresponds to the image is cluster center 2.
  3. Repeat Step 2 to find cluster center 3. This time, find the distances of the remaining images from cluster center 2.
  4. Repeat the process until you have the  $K$  cluster centers.

To train a model in SageMaker, you create a training job. In the request, you provide configuration information by specifying the following `HyperParameters` string maps:

- To specify the number of clusters to create, add the `k` string.
- For greater accuracy, add the optional `extra_center_factor` string.
- To specify the strategy that you want to use to determine the initial cluster centers, add the `init_method` string and set its value to `random` or `k-means++`.

For more information about the SageMaker k-means estimator, see [K-means in the Amazon SageMaker Python SDK documentation](#).

You now have an initial set of cluster centers.

## Step 2: Iterate over the Training Dataset and Calculate Cluster Centers

The cluster centers that you created in the preceding step are mostly random, with some consideration for the training dataset. In this step, you use the training dataset to move these centers toward the true cluster centers. The algorithm iterates over the training dataset, and recalculates the  $K$  cluster centers.

1. Read a mini-batch of observations (a small, randomly chosen subset of all records) from the training dataset and do the following.

### Note

When creating a model training job, you specify the batch size in the `mini_batch_size` string in the `HyperParameters` string map.

- a. Assign all of the observations in the mini-batch to one of the clusters with the closest cluster center.
- b. Calculate the number of observations assigned to each cluster. Then, calculate the proportion of new points assigned per cluster.

For example, consider the following clusters:

Cluster c1 = 100 previously assigned points. You added 25 points from the mini-batch in this step.

Cluster c2 = 150 previously assigned points. You added 40 points from the mini-batch in this step.

Cluster c3 = 450 previously assigned points. You added 5 points from the mini-batch in this step.

Calculate the proportion of new points assigned to each of clusters as follows:

```
p1 = proportion of points assigned to c1 = 25/(100+25)
p2 = proportion of points assigned to c2 = 40/(150+40)
p3 = proportion of points assigned to c3 = 5/(450+5)
```

- c. Compute the center of the new points added to each cluster:

```
d1 = center of the new points added to cluster 1
d2 = center of the new points added to cluster 2
d3 = center of the new points added to cluster 3
```

- d. Compute the weighted average to find the updated cluster centers as follows:

```
Center of cluster 1 = ((1 - p1) * center of cluster 1) + (p1 * d1)
Center of cluster 2 = ((1 - p2) * center of cluster 2) + (p2 * d2)
Center of cluster 3 = ((1 - p3) * center of cluster 3) + (p3 * d3)
```

2. Read the next mini-batch, and repeat Step 1 to recalculate the cluster centers.
3. For more information about mini-batch k-means, see [Web-Scale k-means Clustering](#).

### Step 3: Reduce the Clusters from K to k

If the algorithm created  $K$  clusters—( $K = k*x$ ) where  $x$  is greater than 1—then it reduces the  $K$  clusters to  $k$  clusters. (For more information, see `extra_center_factor` in the preceding discussion.) It does this by applying Lloyd's method with `kmeans++` initialization to the  $K$  cluster centers. For more information about Lloyd's method, see [k-means clustering](#).

### K-Means Hyperparameters

In the `CreateTrainingJob` request, you specify the training algorithm that you want to use. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the k-means training algorithm provided by Amazon SageMaker. For more information about how k-means clustering works, see [How K-Means Clustering Works \(p. 1421\)](#).

Parameter Name	Description
<code>feature_dim</code>	The number of features in the input data. <b>Required</b>

Parameter Name	Description
	Valid values: Positive integer
<b>k</b>	<p>The number of required clusters.</p> <p><b>Required</b></p> <p>Valid values: Positive integer</p>
<b>epochs</b>	<p>The number of passes done over the training data.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 1</p>
<b>eval_metrics</b>	<p>A JSON list of metric types used to report a score for the model. Allowed values are <code>msd</code> for Means Square Error and <code>ssd</code> for Sum of Square Distance. If test data is provided, the score is reported for each of the metrics requested.</p> <p><b>Optional</b></p> <p>Valid values: Either <code>[ \"msd\" ]</code> or <code>[ \"ssd\" ]</code> or <code>[ \"msd\", \"ssd\" ]</code>.</p> <p>Default value: <code>[ \"msd\" ]</code></p>
<b>extra_center_factor</b>	<p>The algorithm creates K centers = <code>num_clusters * extra_center_factor</code> as it runs and reduces the number of centers from K to <code>k</code> when finalizing the model.</p> <p><b>Optional</b></p> <p>Valid values: Either a positive integer or <code>auto</code>.</p> <p>Default value: <code>auto</code></p>
<b>half_life_time_size</b>	<p>Used to determine the weight given to an observation when computing a cluster mean. This weight decays exponentially as more points are observed. When a point is first observed, it is assigned a weight of 1 when computing the cluster mean. The decay constant for the exponential decay function is chosen so that after observing <code>half_life_time_size</code> points, its weight is 1/2. If set to 0, there is no decay.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative integer</p> <p>Default value: 0</p>

Parameter Name	Description
<code>init_method</code>	<p>Method by which the algorithm chooses the initial cluster centers. The standard k-means approach chooses them at random. An alternative k-means++ method chooses the first cluster center at random. Then it spreads out the position of the remaining initial clusters by weighting the selection of centers with a probability distribution that is proportional to the square of the distance of the remaining data points from existing centers.</p> <p><b>Optional</b></p> <p>Valid values: Either <code>random</code> or <code>kmeans++</code>.</p> <p>Default value: <code>random</code></p>
<code>local_lloyd_init_method</code>	<p>The initialization method for Lloyd's expectation-maximization (EM) procedure used to build the final model containing <code>k</code> centers.</p> <p><b>Optional</b></p> <p>Valid values: Either <code>random</code> or <code>kmeans++</code>.</p> <p>Default value: <code>kmeans++</code></p>
<code>local_lloyd_max_iter</code>	<p>The maximum number of iterations for Lloyd's expectation-maximization (EM) procedure used to build the final model containing <code>k</code> centers.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 300</p>
<code>local_lloyd_num_trials</code>	<p>The number of times the Lloyd's expectation-maximization (EM) procedure with the least loss is run when building the final model containing <code>k</code> centers.</p> <p><b>Optional</b></p> <p>Valid values: Either a positive integer or <code>auto</code>.</p> <p>Default value: <code>auto</code></p>
<code>local_lloyd_tol</code>	<p>The tolerance for change in loss for early stopping of Lloyd's expectation-maximization (EM) procedure used to build the final model containing <code>k</code> centers.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range in [0, 1].</p> <p>Default value: 0.0001</p>

Parameter Name	Description
mini_batch_size	<p>The number of observations per mini-batch for the data iterator.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 5000</p>

## Tune a K-Means Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

The Amazon SageMaker k-means algorithm is an unsupervised algorithm that groups data into clusters whose members are as similar as possible. Because it is unsupervised, it doesn't use a validation dataset that hyperparameters can optimize against. But it does take a test dataset and emits metrics that depend on the squared distance between the data points and the final cluster centroids at the end of each training run. To find the model that reports the tightest clusters on the test dataset, you can use a hyperparameter tuning job. The clusters optimize the similarity of their members.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics Computed by the K-Means Algorithm

The k-means algorithm computes the following metrics during training. When tuning a model, choose one of these metrics as the objective metric.

Metric Name	Description	Optimization Direction
test:msd	Mean squared distances between each record in the test set and the closest center of the model.	Minimize
test:ssd	Sum of the squared distances between each record in the test set and the closest center of the model.	Minimize

### Tunable K-Means Hyperparameters

Tune the Amazon SageMaker k-means model with the following hyperparameters. The hyperparameters that have the greatest impact on k-means objective metrics are: `mini_batch_size`, `extra_center_factor`, and `init_method`. Tuning the hyperparameter `epochs` generally results in minor improvements.

Parameter Name	Parameter Type	Recommended Ranges
epochs	IntegerParameterRanges	MinValue: 1, MaxValue:10

Parameter Name	Parameter Type	Recommended Ranges
extra_center_factor	IntegerParameterRanges	MinValue: 4, MaxValue:10
init_method	CategoricalParameterRanges	['kmeans++', 'random']
mini_batch_size	IntegerParameterRanges	MinValue: 3000, MaxValue:15000

## K-Means Response Formats

All SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats - Inference](#). This topic contains a list of the available output formats for the SageMaker k-means algorithm.

### JSON Response Format

```
{
  "predictions": [
    {
      "closest_cluster": 1.0,
      "distance_to_cluster": 3.0,
    },
    {
      "closest_cluster": 2.0,
      "distance_to_cluster": 5.0,
    },
    ....
  ]
}
```

### JSONLINES Response Format

```
{"closest_cluster": 1.0, "distance_to_cluster": 3.0}
{"closest_cluster": 2.0, "distance_to_cluster": 5.0}
```

### RECORDIO Response Format

```
[
  Record = {
    features = {},
    label = {
      'closest_cluster': {
        keys: [],
        values: [1.0, 2.0] # float32
      },
      'distance_to_cluster': {
        keys: [],
        values: [3.0, 5.0] # float32
      },
    }
  }
]
```

### CSV Response Format

The first value in each line corresponds to `closest_cluster`.

The second value in each line corresponds to `distance_to_cluster`.

```
1.0,3.0  
2.0,5.0
```

## K-Nearest Neighbors (k-NN) Algorithm

Amazon SageMaker k-nearest neighbors (k-NN) algorithm is an index-based algorithm. It uses a non-parametric method for classification or regression. For classification problems, the algorithm queries the  $k$  points that are closest to the sample point and returns the most frequently used label of their class as the predicted label. For regression problems, the algorithm queries the  $k$  closest points to the sample point and returns the average of their feature values as the predicted value.

Training with the k-NN algorithm has three steps: sampling, dimension reduction, and index building. Sampling reduces the size of the initial dataset so that it fits into memory. For dimension reduction, the algorithm decreases the feature dimension of the data to reduce the footprint of the k-NN model in memory and inference latency. We provide two methods of dimension reduction methods: random projection and the fast Johnson-Lindenstrauss transform. Typically, you use dimension reduction for high-dimensional ( $d > 1000$ ) datasets to avoid the “curse of dimensionality” that troubles the statistical analysis of data that becomes sparse as dimensionality increases. The main objective of k-NN’s training is to construct the index. The index enables efficient lookups of distances between points whose values or class labels have not yet been determined and the  $k$  nearest points to use for inference.

### Topics

- [Input/Output Interface for the k-NN Algorithm \(p. 1428\)](#)
- [k-NN Sample Notebooks \(p. 1429\)](#)
- [How the k-NN Algorithm Works \(p. 1429\)](#)
- [EC2 Instance Recommendation for the k-NN Algorithm \(p. 1430\)](#)
- [k-NN Hyperparameters \(p. 1430\)](#)
- [Tune a k-NN Model \(p. 1432\)](#)
- [Data Formats for k-NN Training Input \(p. 1433\)](#)
- [k-NN Request and Response Formats \(p. 1434\)](#)

## Input/Output Interface for the k-NN Algorithm

SageMaker k-NN supports train and test data channels.

- Use a *train channel* for data that you want to sample and construct into the k-NN index.
- Use a *test channel* to emit scores in log files. Scores are listed as one line per mini-batch: accuracy for classifier, mean-squared error (mse) for regressor for score.

For training inputs, k-NN supports `text/csv` and `application/x-recordio-protobuf` data formats. For input type `text/csv`, the first `label_size` columns are interpreted as the label vector for that row. You can use either File mode or Pipe mode to train models on data that is formatted as `recordIO-wrapped-protobuf` or as `CSV`.

For inference inputs, k-NN supports the `application/json`, `application/x-recordio-protobuf`, and `text/csv` data formats. The `text/csv` format accepts a `label_size` and `encoding` parameter. It assumes a `label_size` of 0 and a UTF-8 encoding.

For inference outputs, k-NN supports the `application/json` and `application/x-recordio-protobuf` data formats. These two data formats also support a verbose output mode. In verbose output mode, the API provides the search results with the distances vector sorted from smallest to largest, and corresponding elements in the labels vector.

For batch transform, k-NN supports the application/jsonlines data format for both input and output. An example input is as follows:

```
content-type: application/jsonlines

{"features": [1.5, 16.0, 14.0, 23.0]}
{"data": {"features": {"values": [1.5, 16.0, 14.0, 23.0]}}}
```

An example output is as follows:

```
accept: application/jsonlines

{"predicted_label": 0.0}
{"predicted_label": 2.0}
```

For more information on input and output file formats, see [Data Formats for k-NN Training Input \(p. 1433\)](#) for training, [k-NN Request and Response Formats \(p. 1434\)](#) for inference, and the [k-NN Sample Notebooks \(p. 1429\)](#).

## k-NN Sample Notebooks

For a sample notebook that uses the SageMaker k-nearest neighbor algorithm to predict wilderness cover types from geological and forest service data, see the [K-Nearest Neighbor Covertype](#).

Use a Jupyter notebook instance to run the example in SageMaker. To learn how to create and open a Jupyter notebook instance in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the SageMaker example notebooks. Find K-Nearest Neighbor notebooks in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

## How the k-NN Algorithm Works

### Step 1: Sample

To specify the total number of data points to be sampled from the training dataset, use the `sample_size` parameter. For example, if the initial dataset has 1,000 data points and the `sample_size` is set to 100, where the total number of instances is 2, each worker would sample 50 points. A total set of 100 data points would be collected. Sampling runs in linear time with respect to the number of data points.

### Step 2: Perform Dimension Reduction

The current implementation of the k-NN algorithm has two methods of dimension reduction. You specify the method in the `dimension_reduction_type` hyperparameter. The `sign` method specifies a random projection, which uses a linear projection using a matrix of random signs, and the `fjlt` method specifies a fast Johnson-Lindenstrauss transform, a method based on the Fourier transform. Both methods preserve the L2 and inner product distances. The `fjlt` method should be used when the target dimension is large and has better performance with CPU inference. The methods differ in their computational complexity. The `sign` method requires  $O(ndk)$  time to reduce the dimension of a batch of  $n$  points of dimension  $d$  into a target dimension  $k$ . The `fjlt` method requires  $O(nd \log(d))$  time, but the constants involved are larger. Using dimension reduction introduces noise into the data and this noise can reduce prediction accuracy.

### Step 3: Build an Index

During inference, the algorithm queries the index for the k-nearest-neighbors of a sample point. Based on the references to the points, the algorithm makes the classification or regression prediction. It makes the prediction based on the class labels or values provided. k-NN provides three different types of indexes: a flat index, an inverted index, and an inverted index with product quantization. You specify the type with the `index_type` parameter.

## Serialize the Model

When the k-NN algorithm finishes training, it serializes three files to prepare for inference.

- `model_algo-1`: Contains the serialized index for computing the nearest neighbors.
- `model_algo-1.labels`: Contains serialized labels (np.float32 binary format) for computing the predicted label based on the query result from the index.
- `model_algo-1.json`: Contains the JSON-formatted model metadata which stores the `k` and `predictor_type` hyper-parameters from training for inference along with other relevant state.

With the current implementation of k-NN, you can modify the metadata file to change the way predictions are computed. For example, you can change `k` to 10 or change `predictor_type` to `regressor`.

```
{
  "k": 5,
  "predictor_type": "classifier",
  "dimension_reduction": {"type": "sign", "seed": 3, "target_dim": 10, "input_dim": 20},
  "normalize": False,
  "version": "1.0"
}
```

## EC2 Instance Recommendation for the k-NN Algorithm

### Instance Recommendation for Training with the k-NN Algorithm

To start, try running training on a CPU, using, for example, an `ml.m5.2xlarge` instance, or on a GPU using, for example, an `ml.p2.xlarge` instance.

### Instance Recommendation for Inference with the k-NN Algorithm

Inference requests from CPUs generally have a lower average latency than requests from GPUs because there is a tax on CPU-to-GPU communication when you use GPU hardware. However, GPUs generally have higher throughput for larger batches.

## k-NN Hyperparameters

Parameter Name	Description
<code>feature_dim</code>	The number of features in the input data. <b>Required</b> Valid values: positive integer.
<code>k</code>	The number of nearest neighbors. <b>Required</b> Valid values: positive integer
<code>predictor_type</code>	The type of inference to use on the data labels. <b>Required</b> Valid values: <code>classifier</code> for classification or <code>regressor</code> for regression.
<code>sample_size</code>	The number of data points to be sampled from the training data set.

Parameter Name	Description
	<p><b>Required</b></p> <p>Valid values: positive integer</p>
dimension_reduction_target	<p>The target dimension to reduce to.</p> <p><b>Required</b> when you specify the dimension_reduction_type parameter.</p> <p>Valid values: positive integer greater than 0 and less than feature_dim.</p>
dimension_reduction_type	<p>The type of dimension reduction method.</p> <p><b>Optional</b></p> <p>Valid values: <i>sign</i> for random projection or <i>fjlt</i> for the fast Johnson-Lindenstrauss transform.</p> <p>Default value: No dimension reduction</p>
faiss_index_ivf_nlists	<p>The number of centroids to construct in the index when index_type is <i>faiss.IVFFlat</i> or <i>faiss.IVFPQ</i>.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: <i>auto</i>, which resolves to <code>sqrt(sample_size)</code>.</p>
faiss_index_pq_m	<p>The number of vector sub-components to construct in the index when index_type is set to <i>faiss.IVFPQ</i>.</p> <p>The FaceBook AI Similarity Search (FAISS) library requires that the value of faiss_index_pq_m is a divisor of the data dimension. If faiss_index_pq_m is not a divisor of the data dimension, we increase the data dimension to smallest integer divisible by faiss_index_pq_m. If no dimension reduction is applied, the algorithm adds a padding of zeros. If dimension reduction is applied, the algorithm increase the value of the dimension_reduction_target hyper-parameter.</p> <p><b>Optional</b></p> <p>Valid values: One of the following positive integers: 1, 2, 3, 4, 8, 12, 16, 20, 24, 28, 32, 40, 48, 56, 64, 96</p>
index_metric	<p>The metric to measure the distance between points when finding nearest neighbors. When training with index_type set to <i>faiss.IVFPQ</i>, the INNER_PRODUCT distance and COSINE similarity are not supported.</p> <p><b>Optional</b></p> <p>Valid values: <i>L2</i> for Euclidean-distance, <i>INNER_PRODUCT</i> for inner-product distance, <i>COSINE</i> for cosine similarity.</p> <p>Default value: <i>L2</i></p>

Parameter Name	Description
index_type	<p>The type of index.</p> <p><b>Optional</b></p> <p>Valid values: <i>faiss.Flat</i>, <i>faiss.IVFFlat</i>, <i>faiss.IVFPQ</i>.</p> <p>Default values: <i>faiss.Flat</i></p>
mini_batch_size	<p>The number of observations per mini-batch for the data iterator.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 5000</p>

## Tune a k-NN Model

The Amazon SageMaker k-nearest neighbors algorithm is a supervised algorithm. The algorithm consumes a test data set and emits a metric about the accuracy for a classification task or about the mean squared error for a regression task. These accuracy metrics compare the model predictions for their respective task to the ground truth provided by the empirical test data. To find the best model that reports the highest accuracy or lowest error on the test dataset, run a hyperparameter tuning job for k-NN.

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric appropriate for the prediction task of the algorithm. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric. The hyperparameters are used only to help estimate model parameters and are not used by the trained model to make predictions.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics Computed by the k-NN Algorithm

The k-nearest neighbors algorithm computes one of two metrics in the following table during training depending on the type of task specified by the `predictor_type` hyper-parameter.

- *classifier* specifies a classification task and computes `test:accuracy`
- *regressor* specifies a regression task and computes `test:mse`.

Choose the `predictor_type` value appropriate for the type of task undertaken to calculate the relevant objective metric when tuning a model.

Metric Name	Description	Optimization Direction
<code>test:accuracy</code>	When <code>predictor_type</code> is set to <i>classifier</i> , k-NN compares the predicted label, based on the average of the k-nearest neighbors' labels, to the ground truth label provided in the test channel data. The accuracy reported ranges from 0.0 (0%) to 1.0 (100%).	Maximize

Metric Name	Description	Optimization Direction
test:mse	When <code>predictor_type</code> is set to <code>regressor</code> , k-NN compares the predicted label, based on the average of the k-nearest neighbors' labels, to the ground truth label provided in the test channel data. The mean squared error is computed by comparing the two labels.	Minimize

## Tunable k-NN Hyperparameters

Tune the Amazon SageMaker k-nearest neighbor model with the following hyperparameters.

Parameter Name	Parameter Type	Recommended Ranges
k	IntegerParameterRanges	MinValue: 1, MaxValue: 1024
sample_size	IntegerParameterRanges	MinValue: 256, MaxValue: 20000000

## Data Formats for k-NN Training Input

All Amazon SageMaker built-in algorithms adhere to the common input training formats described in [Common Data Formats - Training](#). This topic contains a list of the available input formats for the SageMaker k-nearest-neighbor algorithm.

### CSV Data Format

`content-type: text/csv; label_size=1`

```
4,1.2,1.3,9.6,20.3
```

The first `label_size` columns are interpreted as the label vector for that row.

### RECORDIO Data Format

`content-type: application/x-recordio-protobuf`

```
[  
    Record = {  
        features = {  
            'values': {  
                values: [1.2, 1.3, 9.6, 20.3]  # float32  
            }  
        },  
        label = {  
            'values': {  
                values: [4]  # float32  
            }  
        }  
    }  
]
```

## k-NN Request and Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats - Inference](#). This topic contains a list of the available output formats for the SageMaker k-nearest-neighbor algorithm.

### INPUT: CSV Request Format

content-type: text/csv

```
1.2,1.3,9.6,20.3
```

This accepts a `label_size` or `encoding` parameter. It assumes a `label_size` of 0 and a utf-8 encoding.

### INPUT: JSON Request Format

content-type: application/json

```
{
  "instances": [
    {"data": {"features": {"values": [-3, -1, -4, 2]}},
     {"features": [3.0, 0.1, 0.04, 0.002]}}
  ]
}
```

### INPUT: JSONLINES Request Format

content-type: application/jsonlines

```
{"features": [1.5, 16.0, 14.0, 23.0]}
{"data": {"features": {"values": [1.5, 16.0, 14.0, 23.0]}}}
```

### INPUT: RECORDIO Request Format

content-type: application/x-recordio-protobuf

```
[
  Record = {
    features = {
      'values': {
        values: [-3, -1, -4, 2] # float32
      }
    },
    label = {}
  },
  Record = {
    features = {
      'values': {
        values: [3.0, 0.1, 0.04, 0.002] # float32
      }
    },
    label = {}
  },
]
```

### OUTPUT: JSON Response Format

accept: application/json

```
{
  "predictions": [
```

```

        {"predicted_label": 0.0},
        {"predicted_label": 2.0}
    ]
}
```

### OUTPUT: JSONLINES Response Format

accept: application/jsonlines

```
{"predicted_label": 0.0}
{"predicted_label": 2.0}
```

### OUTPUT: VERBOSE JSON Response Format

In verbose mode, the API provides the search results with the distances vector sorted from smallest to largest, with corresponding elements in the labels vector. In this example, k is set to 3.

accept: application/json; verbose=true

```
{
  "predictions": [
    {
      "predicted_label": 0.0,
      "distances": [3.11792408, 3.89746071, 6.32548437],
      "labels": [0.0, 1.0, 0.0]
    },
    {
      "predicted_label": 2.0,
      "distances": [1.08470316, 3.04917915, 5.25393973],
      "labels": [2.0, 2.0, 0.0]
    }
  ]
}
```

### OUTPUT: RECORDIO-PROTOBUF Response Format

content-type: application/x-recordio-protobuf

```
[
  Record = {
    features = {},
    label = {
      'predicted_label': {
        values: [0.0] # float32
      }
    }
  },
  Record = {
    features = {},
    label = {
      'predicted_label': {
        values: [2.0] # float32
      }
    }
  }
]
```

### OUTPUT: VERBOSE RECORDIO-PROTOBUF Response Format

In verbose mode, the API provides the search results with the distances vector sorted from smallest to largest, with corresponding elements in the labels vector. In this example, k is set to 3.

accept: application/x-recordio-protobuf; verbose=true

```
[  
    Record = {  
        features = {},  
        label = {  
            'predicted_label': {  
                values: [0.0] # float32  
            },  
            'distances': {  
                values: [3.11792408, 3.89746071, 6.32548437] # float32  
            },  
            'labels': {  
                values: [0.0, 1.0, 0.0] # float32  
            }  
        }  
    },  
    Record = {  
        features = {},  
        label = {  
            'predicted_label': {  
                values: [0.0] # float32  
            },  
            'distances': {  
                values: [1.08470316, 3.04917915, 5.25393973] # float32  
            },  
            'labels': {  
                values: [2.0, 2.0, 0.0] # float32  
            }  
        }  
    }  
]
```

### SAMPLE OUTPUT for the k-NN Algorithm

For regressor tasks:

```
[06/08/2018 20:15:33 INFO 140026520049408] #test_score (algo-1) : ('mse',  
0.01333333333333334)
```

For classifier tasks:

```
[06/08/2018 20:15:46 INFO 140285487171328] #test_score (algo-1) : ('accuracy',  
0.9866666666666669)
```

## Latent Dirichlet Allocation (LDA) Algorithm

The Amazon SageMaker Latent Dirichlet Allocation (LDA) algorithm is an unsupervised learning algorithm that attempts to describe a set of observations as a mixture of distinct categories. LDA is most commonly used to discover a user-specified number of topics shared by documents within a text corpus. Here each observation is a document, the features are the presence (or occurrence count) of each word, and the categories are the topics. Since the method is unsupervised, the topics are not specified up front, and are not guaranteed to align with how a human may naturally categorize documents. The topics are learned as a probability distribution over the words that occur in each document. Each document, in turn, is described as a mixture of topics.

The exact content of two documents with similar topic mixtures will not be the same. But overall, you would expect these documents to more frequently use a shared subset of words, than when compared with a document from a different topic mixture. This allows LDA to discover these word groups and use

them to form topics. As an extremely simple example, given a set of documents where the only words that occur within them are: *eat*, *sleep*, *play*, *meow*, and *bark*, LDA might produce topics like the following:

<b>Topic</b>	<b><i>eat</i></b>	<b><i>sleep</i></b>	<b><i>play</i></b>	<b><i>meow</i></b>	<b><i>bark</i></b>
Topic 1	0.1	0.3	0.2	0.4	0.0
Topic 2	0.2	0.1	0.4	0.0	0.3

You can infer that documents that are more likely to fall into Topic 1 are about cats (who are more likely to *meow* and *sleep*), and documents that fall into Topic 2 are about dogs (who prefer to *play* and *bark*). These topics can be found even though the words dog and cat never appear in any of the texts.

### Topics

- [Choosing between Latent Dirichlet Allocation \(LDA\) and Neural Topic Model \(NTM\) \(p. 1437\)](#)
- [Input/Output Interface for the LDA Algorithm \(p. 1438\)](#)
- [EC2 Instance Recommendation for the LDA Algorithm \(p. 1438\)](#)
- [LDA Sample Notebooks \(p. 1438\)](#)
- [How LDA Works \(p. 1438\)](#)
- [LDA Hyperparameters \(p. 1440\)](#)
- [Tune an LDA Model \(p. 1441\)](#)

## Choosing between Latent Dirichlet Allocation (LDA) and Neural Topic Model (NTM)

Topic models are commonly used to produce topics from corpuses that (1) coherently encapsulate semantic meaning and (2) describe documents well. As such, topic models aim to minimize perplexity and maximize topic coherence.

Perplexity is an intrinsic language modeling evaluation metric that measures the inverse of the geometric mean per-word likelihood in your test data. A lower perplexity score indicates better generalization performance. Research has shown that the likelihood computed per word often does not align to human judgement, and can be entirely non-correlated, thus topic coherence has been introduced. Each inferred topic from your model consists of words, and topic coherence is computed to the top N words for that particular topic from your model. It is often defined as the average or median of the pairwise word-similarity scores of the words in that topic e.g., Pointwise Mutual Information (PMI). A promising model generates coherent topics or topics with high topic coherence scores.

While the objective is to train a topic model that minimizes perplexity and maximizes topic coherence, there is often a tradeoff with both LDA and NTM. Recent research by Amazon, Dinget et al., 2018 has shown that NTM is promising for achieving high topic coherence but LDA trained with collapsed Gibbs sampling achieves better perplexity. There is a tradeoff between perplexity and topic coherence. From a practicality standpoint regarding hardware and compute power, SageMaker NTM hardware is more flexible than LDA and can scale better because NTM can run on CPU and GPU and can be parallelized across multiple GPU instances, whereas LDA only supports single-instance CPU training.

### Topics

- [Input/Output Interface for the LDA Algorithm \(p. 1438\)](#)
- [EC2 Instance Recommendation for the LDA Algorithm \(p. 1438\)](#)
- [LDA Sample Notebooks \(p. 1438\)](#)
- [How LDA Works \(p. 1438\)](#)
- [LDA Hyperparameters \(p. 1440\)](#)
- [Tune an LDA Model \(p. 1441\)](#)

## Input/Output Interface for the LDA Algorithm

LDA expects data to be provided on the train channel, and optionally supports a test channel, which is scored by the final model. LDA supports both `recordIO-wrapped-protobuf` (dense and sparse) and CSV file formats. For CSV, the data must be dense and have dimension equal to *number of records* \* *vocabulary size*. LDA can be trained in File or Pipe mode when using recordIO-wrapped protobuf, but only in File mode for the CSV format.

For inference, `text/csv`, `application/json`, and `application/x-recordio-protobuf` content types are supported. Sparse data can also be passed for `application/json` and `application/x-recordio-protobuf`. LDA inference returns `application/json` or `application/x-recordio-protobuf` *predictions*, which include the `topic_mixture` vector for each observation.

Please see the [LDA Sample Notebooks \(p. 1438\)](#) for more detail on training and inference formats.

## EC2 Instance Recommendation for the LDA Algorithm

LDA currently only supports single-instance CPU training. CPU instances are recommended for hosting/inference.

## LDA Sample Notebooks

For a sample notebook that shows how to train the SageMaker Latent Dirichlet Allocation algorithm on a dataset and then how to deploy the trained model to perform inferences about the topic mixtures in input documents, see the [An Introduction to SageMaker LDA](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

## How LDA Works

Amazon SageMaker LDA is an unsupervised learning algorithm that attempts to describe a set of observations as a mixture of different categories. These categories are themselves a probability distribution over the features. LDA is a generative probability model, which means it attempts to provide a model for the distribution of outputs and inputs based on latent variables. This is opposed to discriminative models, which attempt to learn how inputs map to outputs.

You can use LDA for a variety of tasks, from clustering customers based on product purchases to automatic harmonic analysis in music. However, it is most commonly associated with topic modeling in text corpuses. Observations are referred to as documents. The feature set is referred to as vocabulary. A feature is referred to as a word. And the resulting categories are referred to as topics.

### Note

Lemmatization significantly increases algorithm performance and accuracy. Consider pre-processing any input text data.

An LDA model is defined by two parameters:

- $\alpha$ —A prior estimate on topic probability (in other words, the average frequency that each topic within a given document occurs).
- $\beta$ —a collection of  $k$  topics where each topic is given a probability distribution over the vocabulary used in a document corpus, also called a "topic-word distribution."

LDA is a "bag-of-words" model, which means that the order of words does not matter. LDA is a generative model where each document is generated word-by-word by choosing a topic mixture  $\theta \sim \text{Dirichlet}(\alpha)$ .

For each word in the document:

- Choose a topic  $z \sim \text{Multinomial}(\theta)$
- Choose the corresponding topic-word distribution  $\beta_z$ .
- Draw a word  $w \sim \text{Multinomial}(\beta_z)$ .

When training the model, the goal is to find parameters  $\alpha$  and  $\beta$ , which maximize the probability that the text corpus is generated by the model.

The most popular methods for estimating the LDA model use Gibbs sampling or Expectation Maximization (EM) techniques. The Amazon SageMaker LDA uses tensor spectral decomposition. This provides several advantages:

- **Theoretical guarantees on results.** The standard EM-method is guaranteed to converge only to local optima, which are often of poor quality.
- **Embarrassingly parallelizable.** The work can be trivially divided over input documents in both training and inference. The EM-method and Gibbs Sampling approaches can be parallelized, but not as easily.
- **Fast.** Although the EM-method has low iteration cost it is prone to slow convergence rates. Gibbs Sampling is also subject to slow convergence rates and also requires a large number of samples.

At a high-level, the tensor decomposition algorithm follows this process:

1. The goal is to calculate the spectral decomposition of a  $V \times V \times V$  tensor, which summarizes the moments of the documents in our corpus.  $V$  is vocabulary size (in other words, the number of distinct words in all of the documents). The spectral components of this tensor are the LDA parameters  $\alpha$  and  $\beta$ , which maximize the overall likelihood of the document corpus. However, because vocabulary size tends to be large, this  $V \times V \times V$  tensor is prohibitively large to store in memory.
2. Instead, it uses a  $V \times V$  moment matrix, which is the two-dimensional analog of the tensor from step 1, to find a whitening matrix of dimension  $V \times k$ . This matrix can be used to convert the  $V \times V$  moment matrix into a  $k \times k$  identity matrix.  $k$  is the number of topics in the model.
3. This same whitening matrix can then be used to find a smaller  $k \times k \times k$  tensor. When spectrally decomposed, this tensor has components that have a simple relationship with the components of the  $V \times V \times V$  tensor.
4. *Alternating Least Squares* is used to decompose the smaller  $k \times k \times k$  tensor. This provides a substantial improvement in memory consumption and speed. The parameters  $\alpha$  and  $\beta$  can be found by “unwhitening” these outputs in the spectral decomposition.

After the LDA model's parameters have been found, you can find the topic mixtures for each document. You use stochastic gradient descent to maximize the likelihood function of observing a given topic mixture corresponding to these data.

Topic quality can be improved by increasing the number of topics to look for in training and then filtering out poor quality ones. This is in fact done automatically in SageMaker LDA: 25% more topics are computed and only the ones with largest associated Dirichlet priors are returned. To perform further topic filtering and analysis, you can increase the topic count and modify the resulting LDA model as follows:

```
> import mxnet as mx
> alpha, beta = mx.nd.array.load('model.tar.gz')
> # modify alpha and beta
> mx.nd.save('new_model.tar.gz', [new_alpha, new_beta])
> # upload to S3 and create new SageMaker model using the console
```

For more information about algorithms for LDA and the SageMaker implementation, see the following:

- Animashree Anandkumar, Rong Ge, Daniel Hsu, Sham M Kakade, and Matus Telgarsky. *Tensor Decompositions for Learning Latent Variable Models*. Journal of Machine Learning Research, 15:2773–2832, 2014.
- David M Blei, Andrew Y Ng, and Michael I Jordan. *Latent Dirichlet Allocation*. Journal of Machine Learning Research, 3(Jan):993–1022, 2003.
- Thomas L Griffiths and Mark Steyvers. *Finding Scientific Topics*. Proceedings of the National Academy of Sciences, 101(suppl 1):5228–5235, 2004.
- Tamara G Kolda and Brett W Bader. *Tensor Decompositions and Applications*. SIAM Review, 51(3):455–500, 2009.

## LDA Hyperparameters

In the `CreateTrainingJob` request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the LDA training algorithm provided by Amazon SageMaker. For more information, see [How LDA Works \(p. 1438\)](#).

Parameter Name	Description
<code>num_topics</code>	<p>The number of topics for LDA to find within the data.</p> <p><b>Required</b></p> <p>Valid values: positive integer</p>
<code>feature_dim</code>	<p>The size of the vocabulary of the input document corpus.</p> <p><b>Required</b></p> <p>Valid values: positive integer</p>
<code>mini_batch_size</code>	<p>The total number of documents in the input document corpus.</p> <p><b>Required</b></p> <p>Valid values: positive integer</p>
<code>alpha0</code>	<p>Initial guess for the concentration parameter: the sum of the elements of the Dirichlet prior. Small values are more likely to generate sparse topic mixtures and large values (greater than 1.0) produce more uniform mixtures.</p> <p><b>Optional</b></p> <p>Valid values: Positive float</p> <p>Default value: 1.0</p>
<code>max_restarts</code>	<p>The number of restarts to perform during the Alternating Least Squares (ALS) spectral decomposition phase of the algorithm. Can be used to find better quality local minima at the expense of additional computation, but typically should not be adjusted.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 10</p>

Parameter Name	Description
<code>max_iterations</code>	<p>The maximum number of iterations to perform during the ALS phase of the algorithm. Can be used to find better quality minima at the expense of additional computation, but typically should not be adjusted.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 1000</p>
<code>tol</code>	<p>Target error tolerance for the ALS phase of the algorithm. Can be used to find better quality minima at the expense of additional computation, but typically should not be adjusted.</p> <p><b>Optional</b></p> <p>Valid values: Positive float</p> <p>Default value: 1e-8</p>

## Tune an LDA Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

LDA is an unsupervised topic modeling algorithm that attempts to describe a set of observations (documents) as a mixture of different categories (topics). The “per-word log-likelihood” (PWLL) metric measures the likelihood that a learned set of topics (an LDA model) accurately describes a test document dataset. Larger values of PWLL indicate that the test data is more likely to be described by the LDA model.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics Computed by the LDA Algorithm

The LDA algorithm reports on a single metric during training: `test:pwll`. When tuning a model, choose this metric as the objective metric.

Metric Name	Description	Optimization Direction
<code>test:pwll</code>	Per-word log-likelihood on the test dataset. The likelihood that the test dataset is accurately described by the learned LDA model.	Maximize

### Tunable LDA Hyperparameters

You can tune the following hyperparameters for the LDA algorithm. Both hyperparameters, `alpha0` and `num_topics`, can affect the LDA objective metric (`test:pwll`). If you don't already know the optimal

values for these hyperparameters, which maximize per-word log-likelihood and produce an accurate LDA model, automatic model tuning can help find them.

Parameter Name	Parameter Type	Recommended Ranges
alpha0	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 10
num_topics	IntegerParameterRanges	MinValue: 1, MaxValue: 150

## Linear Learner Algorithm

*Linear models* are supervised learning algorithms used for solving either classification or regression problems. For input, you give the model labeled examples  $(x, y)$ .  $x$  is a high-dimensional vector and  $y$  is a numeric label. For binary classification problems, the label must be either 0 or 1. For multiclass classification problems, the labels must be from 0 to `num_classes` - 1. For regression problems,  $y$  is a real number. The algorithm learns a linear function, or, for classification problems, a linear threshold function, and maps a vector  $x$  to an approximation of the label  $y$ .

The Amazon SageMaker linear learner algorithm provides a solution for both classification and regression problems. With the SageMaker algorithm, you can simultaneously explore different training objectives and choose the best solution from a validation set. You can also explore a large number of models and choose the best. The best model optimizes either of the following:

- Continuous objectives, such as mean square error, cross entropy loss, absolute error.
- Discrete objectives suited for classification, such as F1 measure, precision, recall, or accuracy.

Compared with methods that provide a solution for only continuous objectives, the SageMaker linear learner algorithm provides a significant increase in speed over naive hyperparameter optimization techniques. It is also more convenient.

The linear learner algorithm requires a data matrix, with rows representing the observations, and columns representing the dimensions of the features. It also requires an additional column that contains the labels that match the data points. At a minimum, Amazon SageMaker linear learner requires you to specify input and output data locations, and objective type (classification or regression) as arguments. The feature dimension is also required. For more information, see [CreateTrainingJob](#). You can specify additional parameters in the `HyperParameters` string map of the request body. These parameters control the optimization procedure, or specifics of the objective function that you train on. For example, the number of epochs, regularization, and loss type.

If you're using [Managed Spot Training](#), the linear learner algorithm supports using [checkpoints to take a snapshot of the state of the model](#).

### Topics

- [Input/Output interface for the linear learner algorithm \(p. 1443\)](#)
- [EC2 instance recommendation for the linear learner algorithm \(p. 1443\)](#)
- [Linear learner sample notebooks \(p. 1443\)](#)
- [How linear learner works \(p. 1444\)](#)
- [Linear learner hyperparameters \(p. 1445\)](#)
- [Tune a linear learner model \(p. 1454\)](#)
- [Linear learner response formats \(p. 1456\)](#)

## Input/Output interface for the linear learner algorithm

The Amazon SageMaker linear learner algorithm supports three data channels: train, validation (optional), and test (optional). If you provide validation data, the `S3DataDistributionType` should be `FullyReplicated`. The algorithm logs validation loss at every epoch, and uses a sample of the validation data to calibrate and select the best model. If you don't provide validation data, the algorithm uses a sample of the training data to calibrate and select the model. If you provide test data, the algorithm logs include the test score for the final model.

**For training**, the linear learner algorithm supports both `recordIO-wrapped protobuf` and `csv` formats. For the `application/x-recordio-protobuf` input type, only `Float32` tensors are supported. For the `text/csv` input type, the first column is assumed to be the label, which is the target variable for prediction. You can use either File mode or Pipe mode to train linear learner models on data that is formatted as `recordIO-wrapped-protobuf` or as `CSV`.

**For inference**, the linear learner algorithm supports the `application/json`, `application/x-recordio-protobuf`, and `text/csv` formats. When you make predictions on new data, the format of the response depends on the type of model. **For regression** (`predictor_type='regressor'`), the `score` is the prediction produced by the model. **For classification** (`predictor_type='binary_classifier'` or `predictor_type='multiclass_classifier'`), the model returns a `score` and also a `predicted_label`. The `predicted_label` is the class predicted by the model and the `score` measures the strength of that prediction.

- **For binary classification**, `predicted_label` is 0 or 1, and `score` is a single floating point number that indicates how strongly the algorithm believes that the label should be 1.
- **For multiclass classification**, the `predicted_class` will be an integer from 0 to `num_classes-1`, and `score` will be a list of one floating point number per class.

To interpret the `score` in classification problems, you have to consider the loss function used. If the `loss` hyperparameter value is `logistic` for binary classification or `softmax_loss` for multiclass classification, then the `score` can be interpreted as the probability of the corresponding class. These are the loss values used by the linear learner when the `loss` value is `auto` default value. But if the `loss` is set to `hinge_loss`, then the `score` cannot be interpreted as a probability. This is because hinge loss corresponds to a Support Vector Classifier, which does not produce probability estimates.

For more information on input and output file formats, see [Linear learner response formats \(p. 1456\)](#). For more information on inference formats, and the [Linear learner sample notebooks \(p. 1443\)](#).

## EC2 instance recommendation for the linear learner algorithm

You can train the linear learner algorithm on single- or multi-machine CPU and GPU instances. During testing, we have not found substantial evidence that multi-GPU computers are faster than single-GPU computers. Results can vary, depending on your specific use case.

## Linear learner sample notebooks

The following table outlines a variety of sample notebooks that address different use cases of Amazon SageMaker linear learner algorithm.

Notebook Title	Description
<a href="#">An Introduction with the MNIST dataset</a>	Using the MNIST dataset, we train a binary classifier to predict a single digit.
<a href="#">Predict Breast Cancer</a>	Using UCI's Breast Cancer dataset, we train a model to predict Breast Cancer.

Notebook Title	Description
<a href="#">How to Build a Multiclass Classifier?</a>	Using UCI's Covertype dataset, we demonstrate how to train a multiclass classifier.
<a href="#">How to Build a Machine Learning (ML) Pipeline for Inference?</a>	Using a Scikit-learn container, we demonstrate how to build an end-to-end ML pipeline.

For instructions on how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). After you have created a notebook instance and opened it, choose the **SageMaker Examples** tab to see a list of all of the SageMaker samples. The topic modeling example notebooks using the linear learning algorithm are located in the **Introduction to Amazon algorithms** section. To open a notebook, choose its **Use** tab and choose **Create copy**.

## How linear learner works

There are three steps involved in the implementation of the linear learner algorithm: preprocess, train, and validate. Step 4 provides example code that shows how to deploy a trained model.

### Step 1: Preprocess

Normalization, or feature scaling, is an important preprocessing step for certain loss functions that ensures the model being trained on a dataset does not become dominated by the weight of a single feature. The Amazon SageMaker Linear Learner algorithm has a normalization option to assist with this preprocessing step. If normalization is turned on, the algorithm first goes over a small sample of the data to learn the mean value and standard deviation for each feature and for the label. Each of the features in the full dataset is then shifted to have mean of zero and scaled to have a unit standard deviation.

#### Note

For best results, ensure your data is shuffled before training. Training with unshuffled data may cause training to fail.

You can configure whether the linear learner algorithm normalizes the feature data and the labels using the `normalize_data` and `normalize_label` hyperparameters, respectively. Normalization is enabled by default for both features and labels for regression. Only the features can be normalized for binary classification and this is the default behavior.

### Step 2: Train

With the linear learner algorithm, you train with a distributed implementation of stochastic gradient descent (SGD). You can control the optimization process by choosing the optimization algorithm. For example, you can choose to use Adam, AdaGrad, stochastic gradient descent, or other optimization algorithms. You also specify their hyperparameters, such as momentum, learning rate, and the learning rate schedule. If you aren't sure which algorithm or hyperparameter value to use, choose a default that works for the majority of datasets.

During training, you simultaneously optimize multiple models, each with slightly different objectives. For example, you vary L1 or L2 regularization and try out different optimizer settings.

### Step 3: Validate and set the threshold

When training multiple models in parallel, the models are evaluated against a validation set to select the most optimal model once training is complete. For regression, the most optimal model is the one that achieves the best loss on the validation set. For classification, a sample of the validation set is used to calibrate the classification threshold. The most optimal model selected is the one that achieves the best binary classification selection criteria on the validation set. Examples of such criteria include the F1 measure, accuracy, and cross-entropy loss.

### Note

If the algorithm is not provided a validation set, then evaluating and selecting the most optimal model is not possible. To take advantage of parallel training and model selection ensure you provide a validation set to the algorithm.

### Step 4: Deploy a trained linear model

Here is an example of Python code that you can use to deploy a model in MXNet that has been trained with SageMaker linear learner

```
import mxnet as mx
import tarfile
t = tarfile.open('model.tar.gz', 'r:gz')
t.extractall()

# Load the mxnet module from the model files
mod = mx.module.Module.load('mx-mod', 0)
# model's weights
mod._arg_params['fc0_weight'].asnumpy().flatten()

# model bias
mod._arg_params['fc0_bias'].asnumpy().flatten()
mod.bind(data_shapes=data_iter.provide_data) #data_iter is an iterator configured with your test data
#perform inference
result = mod.predict(data_iter)
```

### Linear learner hyperparameters

The following table contains the hyperparameters for the linear learner algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order. When a hyperparameter is set to auto, Amazon SageMaker will automatically calculate and set the value of that hyperparameter.

Parameter Name	Description
<code>num_classes</code>	<p>The number of classes for the response variable. The algorithm assumes that classes are labeled 0, ..., <code>num_classes</code> - 1.</p> <p><b>Required</b> when <code>predictor_type</code> is <code>multiclass_classifier</code>. Otherwise, the algorithm ignores it.</p> <p>Valid values: Integers from 3 to 1,000,000</p>
<code>predictor_type</code>	<p>Specifies the type of target variable as a binary classification, multiclass classification, or regression.</p> <p><b>Required</b></p> <p>Valid values: <code>binary_classifier</code>, <code>multiclass_classifier</code>, or <code>regressor</code></p>
<code>accuracy_top_k</code>	<p>When computing the top-k accuracy metric for multiclass classification, the value of <code>k</code>. If the model assigns one of the top-k scores to the true label, an example is scored as correct.</p> <p><b>Optional</b></p> <p>Valid values: Positive integers</p>

Parameter Name	Description
	Default value: 3
balance_multiclass_weight	<p>Specifies whether to use class weights, which give each class equal importance in the loss function. Used only when the predictor_type is multiclass_classifier.</p> <p><b>Optional</b></p> <p>Valid values: true, false</p> <p>Default value: false</p>
beta_1	<p>The exponential decay rate for first-moment estimates. Applies only when the optimizer value is adam.</p> <p><b>Optional</b></p> <p>Valid values: auto or floating-point value between 0 and 1.0</p> <p>Default value: auto</p>
beta_2	<p>The exponential decay rate for second-moment estimates. Applies only when the optimizer value is adam.</p> <p><b>Optional</b></p> <p>Valid values: auto or floating-point integer between 0 and 1.0</p> <p>Default value: auto</p>
bias_lr_mult	<p>Allows a different learning rate for the bias term. The actual learning rate for the bias is learning_rate * bias_lr_mult.</p> <p><b>Optional</b></p> <p>Valid values: auto or positive floating-point integer</p> <p>Default value: auto</p>
bias_wd_mult	<p>Allows different regularization for the bias term. The actual L2 regularization weight for the bias is wd * bias_wd_mult. By default, there is no regularization on the bias term.</p> <p><b>Optional</b></p> <p>Valid values: auto or non-negative floating-point integer</p> <p>Default value: auto</p>

Parameter Name	Description
binary_classifier_model_selection_criteria	<p>When <code>prediction_type</code> is set to <code>binary_classifier</code>, the model evaluation criteria for the validation dataset (or for the training dataset if you don't provide a validation dataset). Criteria include:</p> <ul style="list-style-type: none"> <li>• <code>accuracy</code>—The model with the highest accuracy.</li> <li>• <code>f_beta</code>—The model with the highest F1 score. The default is <code>F1</code>.</li> <li>• <code>precision_at_target_recall</code>—The model with the highest precision at a given recall target.</li> <li>• <code>recall_at_target_precision</code>—The model with the highest recall at a given precision target.</li> <li>• <code>loss_function</code>—The model with the lowest value of the loss function used in training.</li> </ul> <p><b>Optional</b></p> <p>Valid values: <code>accuracy</code>, <code>f_beta</code>, <code>precision_at_target_recall</code>, <code>recall_at_target_precision</code>, or <code>loss_function</code></p> <p>Default value: <code>accuracy</code></p>
early_stopping_patience	<p>If no improvement is made in the relevant metric, the number of epochs to wait before ending training. If you have provided a value for <code>binary_classifier_model_selection_criteria</code>, the metric is that value. Otherwise, the metric is the same as the value specified for the <code>loss</code> hyperparameter.</p> <p>The metric is evaluated on the validation data. If you haven't provided validation data, the metric is always the same as the value specified for the <code>loss</code> hyperparameter and is evaluated on the training data. To disable early stopping, set <code>early_stopping_patience</code> to a value greater than the value specified for <code>epochs</code>.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 3</p>
early_stopping_tolerance	<p>The relative tolerance to measure an improvement in loss. If the ratio of the improvement in loss divided by the previous best loss is smaller than this value, early stopping considers the improvement to be zero.</p> <p><b>Optional</b></p> <p>Valid values: Positive floating-point integer</p> <p>Default value: 0.001</p>
epochs	<p>The maximum number of passes over the training data.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 15</p>

Parameter Name	Description
<code>f_beta</code>	<p>The value of beta to use when calculating F score metrics for binary or multiclass classification. Also used if the value specified for <code>binary_classifier_model_selection_criteria</code> is <code>f_beta</code>.</p> <p><b>Optional</b></p> <p>Valid values: Positive floating-point integers</p> <p>Default value: 1.0</p>
<code>feature_dim</code>	<p>The number of features in the input data.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code> or positive integer</p> <p>Default values: <code>auto</code></p>
<code>huber_delta</code>	<p>The parameter for Huber loss. During training and metric evaluation, compute L2 loss for errors smaller than delta and L1 loss for errors larger than delta.</p> <p><b>Optional</b></p> <p>Valid values: Positive floating-point integer</p> <p>Default value: 1.0</p>
<code>init_bias</code>	<p>Initial weight for the bias term.</p> <p><b>Optional</b></p> <p>Valid values: Floating-point integer</p> <p>Default value: 0</p>
<code>init_method</code>	<p>Sets the initial distribution function used for model weights. Functions include:</p> <ul style="list-style-type: none"> <li>• <code>uniform</code>—Uniformly distributed between (-scale, +scale)</li> <li>• <code>normal</code>—Normal distribution, with mean 0 and sigma</li> </ul> <p><b>Optional</b></p> <p>Valid values: <code>uniform</code> or <code>normal</code></p> <p>Default value: <code>uniform</code></p>
<code>init_scale</code>	<p>Scales an initial uniform distribution for model weights. Applies only when the <code>init_method</code> hyperparameter is set to <code>uniform</code>.</p> <p><b>Optional</b></p> <p>Valid values: Positive floating-point integer</p> <p>Default value: 0.07</p>

Parameter Name	Description
<code>init_sigma</code>	<p>The initial standard deviation for the normal distribution. Applies only when the <code>init_method</code> hyperparameter is set to <code>normal</code>.</p> <p><b>Optional</b></p> <p>Valid values: Positive floating-point integer</p> <p>Default value: 0.01</p>
<code>l1</code>	<p>The L1 regularization parameter. If you don't want to use L1 regularization, set the value to 0.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code> or non-negative float</p> <p>Default value: <code>auto</code></p>
<code>learning_rate</code>	<p>The step size used by the optimizer for parameter updates.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code> or positive floating-point integer</p> <p>Default value: <code>auto</code>, whose value depends on the optimizer chosen.</p>
<code>loss</code>	<p>Specifies the loss function.</p> <p>The available loss functions and their default values depend on the value of <code>predictor_type</code>:</p> <ul style="list-style-type: none"> <li>If the <code>predictor_type</code> is set to <code>regressor</code>, the available options are <code>auto</code>, <code>squared_loss</code>, <code>absolute_loss</code>, <code>eps_insensitive_squared_loss</code>, <code>eps_insensitive_absolute_loss</code>, <code>quantile_loss</code>, and <code>huber_loss</code>. The default value for <code>auto</code> is <code>squared_loss</code>.</li> <li>If the <code>predictor_type</code> is set to <code>binary_classifier</code>, the available options are <code>auto</code>, <code>logistic</code>, and <code>hinge_loss</code>. The default value for <code>auto</code> is <code>logistic</code>.</li> <li>If the <code>predictor_type</code> is set to <code>multiclass_classifier</code>, the available options are <code>auto</code> and <code>softmax_loss</code>. The default value for <code>auto</code> is <code>softmax_loss</code>.</li> </ul> <p>Valid values: <code>auto</code>, <code>logistic</code>, <code>squared_loss</code>, <code>absolute_loss</code>, <code>hinge_loss</code>, <code>eps_insensitive_squared_loss</code>, <code>eps_insensitive_absolute_loss</code>, <code>quantile_loss</code>, or <code>huber_loss</code></p> <p><b>Optional</b></p> <p>Default value: <code>auto</code></p>

Parameter Name	Description
loss_insensitivity	<p>The parameter for the epsilon-insensitive loss type. During training and metric evaluation, any error smaller than this value is considered to be zero.</p> <p><b>Optional</b></p> <p>Valid values: Positive floating-point integer</p> <p>Default value: 0.01</p>
lr_scheduler_factor	<p>For every <code>lr_scheduler_step</code> hyperparameter, the learning rate decreases by this quantity. Applies only when the <code>use_lr_scheduler</code> hyperparameter is set to <code>true</code>.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code> or positive floating-point integer between 0 and 1</p> <p>Default value: <code>auto</code></p>
lr_scheduler_minimum_lr	<p>The learning rate never decreases to a value lower than the value set for <code>lr_scheduler_minimum_lr</code>. Applies only when the <code>use_lr_scheduler</code> hyperparameter is set to <code>true</code>.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code> or positive floating-point integer</p> <p>Default value: <code>auto</code></p>
lr_scheduler_step	<p>The number of steps between decreases of the learning rate. Applies only when the <code>use_lr_scheduler</code> hyperparameter is set to <code>true</code>.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code> or positive integer</p> <p>Default value: <code>auto</code></p>
margin	<p>The margin for the <code>hinge_loss</code> function.</p> <p><b>Optional</b></p> <p>Valid values: Positive floating-point integer</p> <p>Default value: 1.0</p>
mini_batch_size	<p>The number of observations per mini-batch for the data iterator.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 1000</p>

Parameter Name	Description
momentum	<p>The momentum of the sgd optimizer.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code> or a floating-point integer between 0 and 1.0</p> <p>Default value: <code>auto</code></p>
normalize_data	<p>Normalizes the feature data before training. Data normalization shifts the data for each feature to have a mean of zero and scales it to have unit standard deviation.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code>, <code>true</code>, or <code>false</code></p> <p>Default value: <code>true</code></p>
normalize_label	<p>Normalizes the label. Label normalization shifts the label to have a mean of zero and scales it to have unit standard deviation.</p> <p>The <code>auto</code> default value normalizes the label for regression problems but does not for classification problems. If you set the <code>normalize_label</code> hyperparameter to <code>true</code> for classification problems, the algorithm ignores it.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code>, <code>true</code>, or <code>false</code></p> <p>Default value: <code>auto</code></p>
num_calibration_samples	<p>The number of observations from the validation dataset to use for model calibration (when finding the best threshold).</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code> or positive integer</p> <p>Default value: <code>auto</code></p>
num_models	<p>The number of models to train in parallel. For the default, <code>auto</code>, the algorithm decides the number of parallel models to train. One model is trained according to the given training parameter (regularization, optimizer, loss), and the rest by close parameters.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code> or positive integer</p> <p>Default values: <code>auto</code></p>

Parameter Name	Description
num_point_for_scaler	<p>The number of data points to use for calculating normalization or unbiaseding of terms.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer</p> <p>Default value: 10,000</p>
optimizer	<p>The optimization algorithm to use.</p> <p><b>Optional</b></p> <p>Valid values:</p> <ul style="list-style-type: none"> <li>• <code>auto</code>—The default value.</li> <li>• <code>sgd</code>—Stochastic gradient descent.</li> <li>• <code>adam</code>—<a href="#">Adaptive momentum estimation</a>.</li> <li>• <code>rmsprop</code>—A gradient-based optimization technique that uses a moving average of squared gradients to normalize the gradient.</li> </ul> <p>Default value: <code>auto</code>. The default setting for <code>auto</code> is <code>adam</code>.</p>
positive_example_weight	<p>The weight assigned to positive examples when training a binary classifier. The weight of negative examples is fixed at 1. If you want the algorithm to choose a weight so that errors in classifying negative vs. positive examples have equal impact on training loss, specify <code>balanced</code>. If you want the algorithm to choose the weight that optimizes performance, specify <code>auto</code>.</p> <p><b>Optional</b></p> <p>Valid values: <code>balanced</code>, <code>auto</code>, or a positive floating-point integer</p> <p>Default value: 1.0</p>
quantile	<p>The quantile for quantile loss. For quantile <math>q</math>, the model attempts to produce predictions so that the value of <code>true_label</code> is greater than the prediction with probability <math>q</math>.</p> <p><b>Optional</b></p> <p>Valid values: Floating-point integer between 0 and 1</p> <p>Default value: 0.5</p>
target_precision	<p>The target precision. If <code>binary_classifier_model_selection_criteria</code> is <code>recall_at_target_precision</code>, then precision is held at this value while recall is maximized.</p> <p><b>Optional</b></p> <p>Valid values: Floating-point integer between 0 and 1.0</p> <p>Default value: 0.8</p>

Parameter Name	Description
target_recall	<p>The target recall. If <code>binary_classifier_model_selection_criteria</code> is <code>precision_at_target_recall</code>, then recall is held at this value while precision is maximized.</p> <p><b>Optional</b></p> <p>Valid values: Floating-point integer between 0 and 1.0</p> <p>Default value: 0.8</p>
unbias_data	<p>Unbiases the features before training so that the mean is 0. By default data is unbiased as the <code>use_bias</code> hyperparameter is set to <code>true</code>.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code>, <code>true</code>, or <code>false</code></p> <p>Default value: <code>auto</code></p>
unbias_label	<p>Unbiases labels before training so that the mean is 0. Applies to regression only if the <code>use_bias</code> hyperparameter is set to <code>true</code>.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code>, <code>true</code>, or <code>false</code></p> <p>Default value: <code>auto</code></p>
use_bias	<p>Specifies whether the model should include a bias term, which is the intercept term in the linear equation.</p> <p><b>Optional</b></p> <p>Valid values: <code>true</code> or <code>false</code></p> <p>Default value: <code>true</code></p>
use_lr_scheduler	<p>Whether to use a scheduler for the learning rate. If you want to use a scheduler, specify <code>true</code>.</p> <p><b>Optional</b></p> <p>Valid values: <code>true</code> or <code>false</code></p> <p>Default value: <code>true</code></p>
wd	<p>The weight decay parameter, also known as the L2 regularization parameter. If you don't want to use L2 regularization, set the value to 0.</p> <p><b>Optional</b></p> <p>Valid values: <code>auto</code> or non-negative floating-point integer</p> <p>Default value: <code>auto</code></p>

## Tune a linear learner model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

The linear learner algorithm also has an internal mechanism for tuning hyperparameters separate from the automatic model tuning feature described here. By default, the linear learner algorithm tunes hyperparameters by training multiple models in parallel. When you use automatic model tuning, the linear learner internal tuning mechanism is turned off automatically. This sets the number of parallel models, `num_models`, to 1. The algorithm ignores any value that you set for `num_models`.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics computed by the linear learner algorithm

The linear learner algorithm reports the metrics in the following table, which are computed during training. Choose one of them as the objective metric. To avoid overfitting, we recommend tuning the model against a validation metric instead of a training metric.

Metric Name	Description	Optimization Direction
<code>test:objective_loss</code>	The mean value of the objective loss function on the test dataset after the model is trained. By default, the loss is logistic loss for binary classification and squared loss for regression. To set the loss to other types, use the <code>loss</code> hyperparameter.	Minimize
<code>test:binary_classifier_accuracy</code>	The accuracy of the final model on the test dataset.	Maximize
<code>test:binary_f_beta</code>	The F_beta score of the final model on the test dataset. By default, it is the F1 score, which is the harmonic mean of precision and recall.	Maximize
<code>test:precision</code>	The precision of the final model on the test dataset. If you choose this metric as the objective, we recommend setting a target recall by setting the <code>binary_classifier_model_selection</code> hyperparameter to <code>precision_at_target_recall</code> and setting the value for the <code>target_recall</code> hyperparameter.	Maximize
<code>test:recall</code>	The recall of the final model on the test dataset. If you choose this metric as the objective, we recommend setting a target precision by setting the <code>binary_classifier_model_selection</code> hyperparameter to <code>recall_at_target_precision</code> and setting the value for the <code>target_precision</code> hyperparameter.	Maximize
<code>validation:objective_loss</code>	The mean value of the objective loss function on the validation dataset every epoch. By default,	Minimize

Metric Name	Description	Optimization Direction
	the loss is logistic loss for binary classification and squared loss for regression. To set loss to other types, use the <code>loss</code> hyperparameter.	
<code>validation:binary_accuracy</code>	The accuracy of the final model on the validation dataset.	Maximize
<code>validation:binary_f1</code>	The F_beta score of the final model on the validation dataset. By default, the F_beta score is the F1 score, which is the harmonic mean of the <code>validation:precision</code> and <code>validation:recall</code> metrics.	Maximize
<code>validation:precision</code>	The precision of the final model on the validation dataset. If you choose this metric as the objective, we recommend setting a target recall by setting the <code>binary_classifier_model_selection</code> hyperparameter to <code>precision_at_target_recall</code> and setting the value for the <code>target_recall</code> hyperparameter.	Maximize
<code>validation:recall</code>	The recall of the final model on the validation dataset. If you choose this metric as the objective, we recommend setting a target precision by setting the <code>binary_classifier_model_selection</code> hyperparameter to <code>recall_at_target_precision</code> and setting the value for the <code>target_precision</code> hyperparameter.	Maximize

## Tuning linear learner hyperparameters

You can tune a linear learner model with the following hyperparameters.

Parameter Name	Parameter Type	Recommended Ranges
<code>wd</code>	<code>ContinuousParameterRanges</code>	<code>MinValue: 1e-7, MaxValue: 1</code>
<code>l1</code>	<code>ContinuousParameterRanges</code>	<code>MinValue: 1e-7, MaxValue: 1</code>
<code>learning_rate</code>	<code>ContinuousParameterRanges</code>	<code>MinValue: 1e-5, MaxValue: 1</code>
<code>mini_batch_size</code>	<code>IntegerParameterRanges</code>	<code>MinValue: 100, MaxValue: 5000</code>
<code>use_bias</code>	<code>CategoricalParameterRanges</code>	<code>[True, False]</code>
<code>positive_example_weight</code>	<code>ContinuousParameterRanges</code>	<code>MinValue: 1e-5, MaxValue: 1e5</code>

## Linear learner response formats

### JSON response formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats - Inference](#). The following are the available output formats for the SageMaker linear learner algorithm.

#### Binary Classification

```
let response = {
  "predictions": [
    {
      "score": 0.4,
      "predicted_label": 0
    }
  ]
}
```

#### Multiclass Classification

```
let response = {
  "predictions": [
    {
      "score": [0.1, 0.2, 0.4, 0.3],
      "predicted_label": 2
    }
  ]
}
```

#### Regression

```
let response = {
  "predictions": [
    {
      "score": 0.4
    }
  ]
}
```

## JSONLINES response formats

### Binary Classification

```
{"score": 0.4, "predicted_label": 0}
```

### Multiclass Classification

```
{"score": [0.1, 0.2, 0.4, 0.3], "predicted_label": 2}
```

### Regression

```
{"score": 0.4}
```

## RECORDIO response formats

### Binary Classification

```
[  
    Record = {  
        features = {},  
        label = {  
            'score': {  
                keys: [],  
                values: [0.4] # float32  
            },  
            'predicted_label': {  
                keys: [],  
                values: [0.0] # float32  
            }  
        }  
    }  
]
```

### Multiclass Classification

```
[  
    Record = {  
        "features": [],  
        "label": {  
            "score": {  
                "values": [0.1, 0.2, 0.3, 0.4]  
            },  
            "predicted_label": {  
                "values": [3]  
            }  
        },  
        "uid": "abc123",  
        "metadata": "{created_at: '2017-06-03'}"  
    }  
]
```

### Regression

```
[  
    Record = {  
        features = {},  
        label = {  
            'score': {  
                keys: [],  
                values: [0.4] # float32  
            }  
        }  
    }  
]
```

## Neural Topic Model (NTM) Algorithm

Amazon SageMaker NTM is an unsupervised learning algorithm that is used to organize a corpus of documents into *topics* that contain word groupings based on their statistical distribution. Documents that contain frequent occurrences of words such as "bike", "car", "train", "mileage", and "speed" are likely to share a topic on "transportation" for example. Topic modeling can be used to classify or summarize documents based on the topics detected or to retrieve information or recommend content based on topic similarities. The topics from documents that NTM learns are characterized as a *latent representation* because the topics are inferred from the observed word distributions in the corpus. The semantics of topics are usually inferred by examining the top ranking words they contain. Because the method is unsupervised, only the number of topics, not the topics themselves, are prespecified. In addition, the topics are not guaranteed to align with how a human might naturally categorize documents.

Topic modeling provides a way to visualize the contents of a large document corpus in terms of the learned topics. Documents relevant to each topic might be indexed or searched for based on their soft topic labels. The latent representations of documents might also be used to find similar documents in the topic space. You can also use the latent representations of documents that the topic model learns for input to another supervised algorithm such as a document classifier. Because the latent representations of documents are expected to capture the semantics of the underlying documents, algorithms based in part on these representations are expected to perform better than those based on lexical features alone.

Although you can use both the Amazon SageMaker NTM and LDA algorithms for topic modeling, they are distinct algorithms and can be expected to produce different results on the same input data.

For more information on the mathematics behind NTM, see [Neural Variational Inference for Text Processing](#).

### Topics

- [Input/Output Interface for the NTM Algorithm \(p. 1458\)](#)
- [EC2 Instance Recommendation for the NTM Algorithm \(p. 1459\)](#)
- [NTM Sample Notebooks \(p. 1459\)](#)
- [NTM Hyperparameters \(p. 1459\)](#)
- [Tune an NTM Model \(p. 1461\)](#)
- [NTM Response Formats \(p. 1462\)](#)

## Input/Output Interface for the NTM Algorithm

Amazon SageMaker Neural Topic Model supports four data channels: train, validation, test, and auxiliary. The validation, test, and auxiliary data channels are optional. If you specify any of these optional channels, set the value of the `S3DataDistributionType` parameter for them to `FullyReplicated`. If you provide validation data, the loss on this data is logged at every epoch, and the model stops training as soon as it detects that the validation loss is not improving. If you don't provide validation data, the algorithm stops early based on the training data, but this can be less efficient. If you provide test data, the algorithm reports the test loss from the final model.

The train, validation, and test data channels for NTM support both `recordIO-wrapped-protobuf` (dense and sparse) and `csv` file formats. For `csv` format, each row must be represented densely with zero counts for words not present in the corresponding document, and have dimension equal to:  $(\text{number of records}) * (\text{vocabulary size})$ . You can use either File mode or Pipe mode to train models on data that is formatted as `recordIO-wrapped-protobuf` or as `csv`. The auxiliary channel is used to supply a text file that contains vocabulary. By supplying the vocabulary file, users are able to see the top words for each of the topics printed in the log instead of their integer IDs. Having the vocabulary file also allows NTM to compute the Word Embedding Topic Coherence (WETC) scores, a new metric displayed in the log that captures similarity among the top words in each topic effectively. The `ContentType` for the auxiliary channel is `text/plain`, with each line containing a single word, in the order corresponding to the integer IDs provided in the data. The vocabulary file must be named `vocab.txt` and currently only UTF-8 encoding is supported.

For inference, `text/csv`, `application/json`, `application/jsonlines`, and `application/x-recordio-protobuf` content types are supported. Sparse data can also be passed for `application/json` and `application/x-recordio-protobuf`. NTM inference returns `application/json` or `application/x-recordio-protobuf` *predictions*, which include the `topic_weights` vector for each observation.

See the [blog post](#) and the companion [notebook](#) for more details on using the auxiliary channel and the WETC scores. For more information on how to compute the WETC score, see [Coherence-Aware Neural Topic Modeling](#). We used the pairwise WETC described in this paper for the Amazon SageMaker Neural Topic Model.

For more information on input and output file formats, see [NTM Response Formats \(p. 1462\)](#) for inference and the [NTM Sample Notebooks \(p. 1459\)](#).

## EC2 Instance Recommendation for the NTM Algorithm

NTM training supports both GPU and CPU instance types. We recommend GPU instances, but for certain workloads, CPU instances may result in lower training costs. CPU instances should be sufficient for inference.

## NTM Sample Notebooks

For a sample notebook that uses the SageMaker NTM algorithm to uncover topics in documents from a synthetic data source where the topic distributions are known, see the [Introduction to Basic Functionality of NTM](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

## NTM Hyperparameters

Parameter Name	Description
<code>feature_dim</code>	The vocabulary size of the dataset. <b>Required</b> Valid values: Positive integer (min: 1, max: 1,000,000)
<code>num_topics</code>	The number of required topics. <b>Required</b> Valid values: Positive integer (min: 2, max: 1000)
<code>batch_norm</code>	Whether to use batch normalization during training. <b>Optional</b> Valid values: <i>true</i> or <i>false</i> Default value: <i>false</i>
<code>clip_gradient</code>	The maximum magnitude for each gradient component. <b>Optional</b> Valid values: Float (min: 1e-3) Default value: Infinity
<code>encoder_layers</code>	The number of layers in the encoder and the output size of each layer. When set to <i>auto</i> , the algorithm uses two layers of sizes 3 x <code>num_topics</code> and 2 x <code>num_topics</code> respectively. <b>Optional</b> Valid values: Comma-separated list of positive integers or <i>auto</i>

Parameter Name	Description
	Default value: <i>auto</i>
encoder_layers_activation	<p>The activation function to use in the encoder layers.</p> <p><b>Optional</b></p> <p>Valid values:</p> <ul style="list-style-type: none"> <li>• sigmoid: <a href="#">Sigmoid function</a></li> <li>• tanh: <a href="#">Hyperbolic tangent</a></li> <li>• relu: <a href="#">Rectified linear unit</a></li> </ul> <p>Default value: sigmoid</p>
epochs	<p>The maximum number of passes over the training data.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer (min: 1)</p> <p>Default value: 50</p>
learning_rate	<p>The learning rate for the optimizer.</p> <p><b>Optional</b></p> <p>Valid values: Float (min: 1e-6, max: 1.0)</p> <p>Default value: 0.001</p>
mini_batch_size	<p>The number of examples in each mini batch.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer (min: 1, max: 10000)</p> <p>Default value: 256</p>
num_patience_epochs	<p>The number of successive epochs over which early stopping criterion is evaluated. Early stopping is triggered when the change in the loss function drops below the specified tolerance within the last num_patience_epochs number of epochs. To disable early stopping, set num_patience_epochs to a value larger than epochs.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer (min: 1)</p> <p>Default value: 3</p>

Parameter Name	Description
optimizer	<p>The optimizer to use for training.</p> <p><b>Optional</b></p> <p>Valid values:</p> <ul style="list-style-type: none"> <li>sgd: <a href="#">Stochastic gradient descent</a></li> <li>adam: <a href="#">Adaptive momentum estimation</a></li> <li>adagrad: <a href="#">Adaptive gradient algorithm</a></li> <li>adadelta: <a href="#">An adaptive learning rate algorithm</a></li> <li>rmsprop: <a href="#">Root mean square propagation</a></li> </ul> <p>Default value: adadelta</p>
rescale_gradient	<p>The rescale factor for gradient.</p> <p><b>Optional</b></p> <p>Valid values: float (min: 1e-3, max: 1.0)</p> <p>Default value: 1.0</p>
sub_sample	<p>The fraction of the training data to sample for training per epoch.</p> <p><b>Optional</b></p> <p>Valid values: Float (min: 0.0, max: 1.0)</p> <p>Default value: 1.0</p>
tolerance	<p>The maximum relative change in the loss function. Early stopping is triggered when change in the loss function drops below this value within the last num_patience_epochs number of epochs.</p> <p><b>Optional</b></p> <p>Valid values: Float (min: 1e-6, max: 0.1)</p> <p>Default value: 0.001</p>
weight_decay	<p>The weight decay coefficient. Adds L2 regularization.</p> <p><b>Optional</b></p> <p>Valid values: Float (min: 0.0, max: 1.0)</p> <p>Default value: 0.0</p>

## Tune an NTM Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

Amazon SageMaker NTM is an unsupervised learning algorithm that learns latent representations of large collections of discrete data, such as a corpus of documents. Latent representations use inferred variables that are not directly measured to model the observations in a dataset. Automatic model tuning on NTM helps you find the model that minimizes loss over the training or validation data. *Training loss* measures how well the model fits the training data. *Validation loss* measures how well the model can generalize to data that it is not trained on. Low training loss indicates that a model is a good fit to the training data. Low validation loss indicates that a model has not overfit the training data and so should be able to model documents successfully on which it has not been trained. Usually, it's preferable to have both losses be small. However, minimizing training loss too much might result in overfitting and increase validation loss, which would reduce the generality of the model.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics Computed by the NTM Algorithm

The NTM algorithm reports a single metric that is computed during training: `validation:total_loss`. The total loss is the sum of the reconstruction loss and Kullback-Leibler divergence. When tuning hyperparameter values, choose this metric as the objective.

Metric Name	Description	Optimization Direction
<code>validation:total_loss</code>	Total Loss on validation set	Minimize

### Tunable NTM Hyperparameters

You can tune the following hyperparameters for the NTM algorithm. Usually setting low `mini_batch_size` and small `learning_rate` values results in lower validation losses, although it might take longer to train. Low validation losses don't necessarily produce more coherent topics as interpreted by humans. The effect of other hyperparameters on training and validation loss can vary from dataset to dataset. To see which values are compatible, see [NTM Hyperparameters \(p. 1459\)](#).

Parameter Name	Parameter Type	Recommended Ranges
<code>encoder_layers_activation</code>	CategoricalParameterRanges	[ <code>'sigmoid'</code> , <code>'tanh'</code> , <code>'relu'</code> ]
<code>learning_rate</code>	ContinuousParameterRange	MinValue: <code>1e-4</code> , MaxValue: <code>0.1</code>
<code>mini_batch_size</code>	IntegerParameterRanges	MinValue: <code>16</code> , MaxValue: <code>2048</code>
<code>optimizer</code>	CategoricalParameterRanges	[ <code>'sgd'</code> , <code>'adam'</code> , <code>'adadelta'</code> ]
<code>rescale_gradient</code>	ContinuousParameterRange	MinValue: <code>0.1</code> , MaxValue: <code>1.0</code>
<code>weight_decay</code>	ContinuousParameterRange	MinValue: <code>0.0</code> , MaxValue: <code>1.0</code>

### NTM Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats - Inference](#). This topic contains a list of the available output formats for the SageMaker NTM algorithm.

## JSON Response Format

```
{  
    "predictions": [  
        {"topic_weights": [0.02, 0.1, 0,...]},  
        {"topic_weights": [0.25, 0.067, 0,...]}  
    ]  
}
```

## JSONLINES Response Format

```
{"topic_weights": [0.02, 0.1, 0,...]}  
{"topic_weights": [0.25, 0.067, 0,...]}
```

## RECORDIO Response Format

```
[  
    Record = {  
        features = {},  
        label = {  
            'topic_weights': {  
                keys: [],  
                values: [0.25, 0.067, 0, ...] # float32  
            }  
        }  
    },  
    Record = {  
        features = {},  
        label = {  
            'topic_weights': {  
                keys: [],  
                values: [0.25, 0.067, 0, ...] # float32  
            }  
        }  
    }  
]
```

## Object2Vec Algorithm

The Amazon SageMaker Object2Vec algorithm is a general-purpose neural embedding algorithm that is highly customizable. It can learn low-dimensional dense embeddings of high-dimensional objects. The embeddings are learned in a way that preserves the semantics of the relationship between pairs of objects in the original space in the embedding space. You can use the learned embeddings to efficiently compute nearest neighbors of objects and to visualize natural clusters of related objects in low-dimensional space, for example. You can also use the embeddings as features of the corresponding objects in downstream supervised tasks, such as classification or regression.

Object2Vec generalizes the well-known Word2Vec embedding technique for words that is optimized in the SageMaker [BlazingText algorithm \(p. 1364\)](#). For a blog post that discusses how to apply Object2Vec to some practical use cases, see [Introduction to Amazon SageMaker Object2Vec](#).

### Topics

- [I/O Interface for the Object2Vec Algorithm \(p. 1464\)](#)
- [EC2 Instance Recommendation for the Object2Vec Algorithm \(p. 1465\)](#)
- [Object2Vec Sample Notebooks \(p. 1465\)](#)
- [How Object2Vec Works \(p. 1465\)](#)

- [Object2Vec Hyperparameters \(p. 1467\)](#)
- [Tune an Object2Vec Model \(p. 1475\)](#)
- [Data Formats for Object2Vec Training \(p. 1477\)](#)
- [Data Formats for Object2Vec Inference \(p. 1477\)](#)
- [Encoder Embeddings for Object2Vec \(p. 1478\)](#)

## I/O Interface for the Object2Vec Algorithm

You can use Object2Vec on many input data types, including the following examples.

Input Data Type	Example
Sentence-sentence pairs	"A soccer game with multiple males playing." and "Some men are playing a sport."
Labels-sequence pairs	The genre tags of the movie "Titanic", such as "Romance" and "Drama", and its short description: "James Cameron's Titanic is an epic, action-packed romance set against the ill-fated maiden voyage of the R.M.S. Titanic. She was the most luxurious liner of her era, a ship of dreams, which ultimately carried over 1,500 people to their death in the ice cold waters of the North Atlantic in the early hours of April 15, 1912."
Customer-customer pairs	The customer ID of Jane and customer ID of Jackie.
Product-product pairs	The product ID of football and product ID of basketball.
Item review user-item pairs	A user's ID and the items she has bought, such as apple, pear, and orange.

To transform the input data into the supported formats, you must preprocess it. Currently, Object2Vec natively supports two types of input:

- A discrete token, which is represented as a list of a single `integer-id`. For example, `[10]`.
- A sequences of discrete tokens, which is represented as a list of `integer-ids`. For example, `[0, 12, 10, 13]`.

The object in each pair can be asymmetric. For example, the pairs can be (token, sequence) or (token, token) or (sequence, sequence). For token inputs, the algorithm supports simple embeddings as compatible encoders. For sequences of token vectors, the algorithm supports the following as encoders:

- Average-pooled embeddings
- Hierarchical convolutional neural networks (CNNs),
- Multi-layered bidirectional long short-term memory (BiLSTMs)

The input label for each pair can be one of the following:

- A categorical label that expresses the relationship between the objects in the pair
- A score that expresses the strength of the similarity between the two objects

For categorical labels used in classification, the algorithm supports the cross-entropy loss function. For ratings/score-based labels used in regression, the algorithm supports the mean squared error (MSE) loss

function. Specify these loss functions with the `output_layer` hyperparameter when you create the model training job.

## EC2 Instance Recommendation for the Object2Vec Algorithm

The type of Amazon Elastic Compute Cloud (Amazon EC2) instance that you use depends on whether you are training or running inferences.

### Instance Recommendation for Training

When training a model using the Object2Vec algorithm on a CPU, start with an `ml.m5.2xlarge` instance. For training on a GPU, start with an `ml.p2.xlarge` instance. If the training takes too long on this instance, you can use a larger instance, such as an `ml.m5.4xlarge` or an `ml.m5.12xlarge` instance. Currently, the Object2Vec algorithm can train only on a single machine. However, it does offer support for multiple GPUs.

### Instance Recommendation for Inference

For inference with a trained Object2Vec model that has a deep neural network, we recommend using `ml.p3.2xlarge` GPU instance. Due to GPU memory scarcity, the `INFERENCE_PREFERRED_MODE` environment variable can be specified to optimize on whether the [the section called "GPU optimization: Classification or Regression" \(p. 1477\)](#) or [the section called "GPU optimization: Encoder Embeddings" \(p. 1478\)](#) inference network is loaded into GPU.

## Object2Vec Sample Notebooks

- [Using Object2Vec to Encode Sentences into Fixed Length Embeddings](#)
- [Using Object2Vec to learn document embeddings](#)

### Note

To run the notebooks on a notebook instance, see [Example Notebooks \(p. 131\)](#). To run the notebooks on Studio, see [Create or Open an Amazon SageMaker Studio Notebook \(p. 79\)](#).

## How Object2Vec Works

When using the Amazon SageMaker Object2Vec algorithm, you follow the standard workflow: process the data, train the model, and produce inferences.

### Topics

- [Step 1: Process Data \(p. 1465\)](#)
- [Step 2: Train a Model \(p. 1465\)](#)
- [Step 3: Produce Inferences \(p. 1466\)](#)

### Step 1: Process Data

During preprocessing, convert the data to the [JSON Lines](#) text file format specified in [Data Formats for Object2Vec Training \(p. 1477\)](#). To get the highest accuracy during training, also randomly shuffle the data before feeding it into the model. How you generate random permutations depends on the language. For python, you could use `np.random.shuffle`; for Unix, `shuf`.

### Step 2: Train a Model

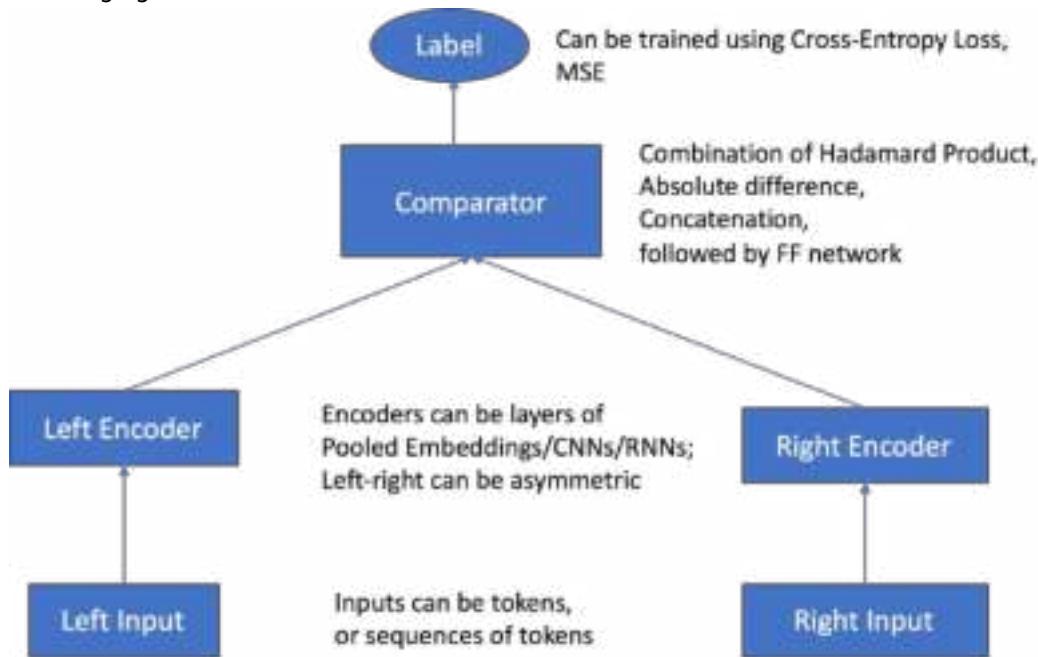
The SageMaker Object2Vec algorithm has the following main components:

- **Two input channels** – The input channels take a pair of objects of the same or different types as inputs, and pass them to independent and customizable encoders.

- **Two encoders** – The two encoders, enc0 and enc1, convert each object into a fixed-length embedding vector. The encoded embeddings of the objects in the pair are then passed into a comparator.
- **A comparator** – The comparator compares the embeddings in different ways and outputs scores that indicate the strength of the relationship between the paired objects. In the output score for a sentence pair. For example, 1 indicates a strong relationship between a sentence pair, and 0 represents a weak relationship.

During training, the algorithm accepts pairs of objects and their relationship labels or scores as inputs. The objects in each pair can be of different types, as described earlier. If the inputs to both encoders are composed of the same token-level units, you can use a shared token embedding layer by setting the `tied_token_embedding_weight` hyperparameter to `True` when you create the training job. This is possible, for example, when comparing sentences that both have word token-level units. To generate negative samples at a specified rate, set the `negative_sampling_rate` hyperparameter to the desired ratio of negative to positive samples. This hyperparameter expedites learning how to discriminate between the positive samples observed in the training data and the negative samples that are not likely to be observed.

Pairs of objects are passed through independent, customizable encoders that are compatible with the input types of corresponding objects. The encoders convert each object in a pair into a fixed-length embedding vector of equal length. The pair of vectors are passed to a comparator operator, which assembles the vectors into a single vector using the value specified in the `comparator_list` hyperparameter. The assembled vector then passes through a multilayer perceptron (MLP) layer, which produces an output that the loss function compares with the labels that you provided. This comparison evaluates the strength of the relationship between the objects in the pair as predicted by the model. The following figure shows this workflow.



Architecture of the Object2Vec Algorithm from Data Inputs to Scores

### Step 3: Produce Inferences

After the model is trained, you can use the trained encoder to preprocess input objects or to perform two types of inference:

- To convert singleton input objects into fixed-length embeddings using the corresponding encoder

- To predict the relationship label or score between a pair of input objects

The inference server automatically figures out which of the types is requested based on the input data. To get the embeddings as output, provide only one input. To predict the relationship label or score, provide both inputs in the pair.

## Object2Vec Hyperparameters

In the `CreateTrainingJob` request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the Object2Vec training algorithm.

Parameter Name	Description
<code>enc0_max_seq_len</code>	<p>The maximum sequence length for the enc0 encoder.</p> <p><b>Required</b></p> <p>Valid values: <math>1 \leq \text{integer} \leq 5000</math></p>
<code>enc0_vocab_size</code>	<p>The vocabulary size of enc0 tokens.</p> <p><b>Required</b></p> <p>Valid values: <math>2 \leq \text{integer} \leq 3000000</math></p>
<code>bucket_width</code>	<p>The allowed difference between data sequence length when bucketing is enabled. To enable bucketing, specify a non-zero value for this parameter.</p> <p><b>Optional</b></p> <p>Valid values: <math>0 \leq \text{integer} \leq 100</math></p> <p>Default value: 0 (no bucketing)</p>
<code>comparator_list</code>	<p>A list used to customize the way in which two embeddings are compared. The Object2Vec comparator operator layer takes the encodings from both encoders as inputs and outputs a single vector. This vector is a concatenation of subvectors. The string values passed to the <code>comparator_list</code> and the order in which they are passed determine how these subvectors are assembled. For example, if <code>comparator_list="hadamard, concat"</code>, then the comparator operator constructs the vector by concatenating the Hadamard product of two encodings and the concatenation of two encodings. If, on the other hand, <code>comparator_list="hadamard"</code>, then the comparator operator constructs the vector as the hadamard product of only two encodings.</p> <p><b>Optional</b></p> <p>Valid values: A string that contains any combination of the names of the three binary operators: <code>hadamard</code>, <code>concat</code>, or <code>abs_diff</code>. The Object2Vec algorithm currently requires that the two vector encodings have the same dimension. These operators produce the subvectors as follows:</p> <ul style="list-style-type: none"> <li>• <code>hadamard</code>: Constructs a vector as the <a href="#">Hadamard (element-wise) product</a> of two encodings.</li> </ul>

Parameter Name	Description
	<ul style="list-style-type: none"> <li>concat: Constructs a vector as the concatenation of two encodings.</li> <li>abs_diff: Constructs a vector as the absolute difference between two encodings.</li> </ul> <p>Default value: "hadamard, concat, abs_diff"</p>
dropout	<p>The dropout probability for network layers. <i>Dropout</i> is a form of regularization used in neural networks that reduces overfitting by trimming codependent neurons.</p> <p><b>Optional</b></p> <p>Valid values: <math>0.0 \leq \text{float} \leq 1.0</math></p> <p>Default value: 0.0</p>
early_stopping_patience	<p>The number of consecutive epochs without improvement allowed before early stopping is applied. Improvement is defined by with the <code>early_stopping_tolerance</code> hyperparameter.</p> <p><b>Optional</b></p> <p>Valid values: <math>1 \leq \text{integer} \leq 5</math></p> <p>Default value: 3</p>
early_stopping_tolerance	<p>The reduction in the loss function that an algorithm must achieve between consecutive epochs to avoid early stopping after the number of consecutive epochs specified in the <code>early_stopping_patience</code> hyperparameter concludes.</p> <p><b>Optional</b></p> <p>Valid values: <math>0.000001 \leq \text{float} \leq 0.1</math></p> <p>Default value: 0.01</p>
enc_dim	<p>The dimension of the output of the embedding layer.</p> <p><b>Optional</b></p> <p>Valid values: <math>4 \leq \text{integer} \leq 10000</math></p> <p>Default value: 4096</p>

Parameter Name	Description
enc0_network	<p>The network model for the enc0 encoder.</p> <p><b>Optional</b></p> <p>Valid values: hcnn, bilstm, or pooled_embedding</p> <ul style="list-style-type: none"> <li>• hcnn: A hierarchical convolutional neural network.</li> <li>• bilstm: A bidirectional long short-term memory network (biLSTM), in which the signal propagates backward and forward in time. This is an appropriate recurrent neural network (RNN) architecture for sequential learning tasks.</li> <li>• pooled_embedding: Averages the embeddings of all of the tokens in the input.</li> </ul> <p>Default value: hcnn</p>
enc0_cnn_filter_width	<p>The filter width of the convolutional neural network (CNN) enc0 encoder.</p> <p><b>Conditional</b></p> <p>Valid values: <math>1 \leq \text{integer} \leq 9</math></p> <p>Default value: 3</p>
enc0_freeze_pretrained_embedding	<p>Whether to freeze enc0 pretrained embedding weights.</p> <p><b>Conditional</b></p> <p>Valid values: True or False</p> <p>Default value: True</p>
enc0_layers	<p>The number of layers in the enc0 encoder.</p> <p><b>Conditional</b></p> <p>Valid values: auto or <math>1 \leq \text{integer} \leq 4</math></p> <ul style="list-style-type: none"> <li>• For hcnn, auto means 4.</li> <li>• For bilstm, auto means 1.</li> <li>• For pooled_embedding, auto ignores the number of layers.</li> </ul> <p>Default value: auto</p>
enc0_pretrained_embedding_file	<p>filename of the pretrained enc0 token embedding file in the auxiliary data channel.</p> <p><b>Conditional</b></p> <p>Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9\.\_\_]</p> <p>Default value: "" (empty string)</p>

Parameter Name	Description
enc0_token_embedding_dim	<p>The output dimension of the enc0 token embedding layer.</p> <p><b>Conditional</b></p> <p>Valid values: <math>2 \leq \text{integer} \leq 1000</math></p> <p>Default value: 300</p>
enc0_vocab_file	<p>The vocabulary file for mapping pretrained enc0 token embedding vectors to numerical vocabulary IDs.</p> <p><b>Conditional</b></p> <p>Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9\.\_]</p> <p>Default value: "" (empty string)</p>
enc1_network	<p>The network model for the enc1 encoder. If you want the enc1 encoder to use the same network model as enc0, including the hyperparameter values, set the value to enc0.</p> <p><b>Note</b> Even when the enc0 and enc1 encoder networks have symmetric architectures, you can't shared parameter values for these networks.</p> <p><b>Optional</b></p> <p>Valid values: enc0, hcnn, bilstm, or pooled_embedding</p> <ul style="list-style-type: none"> <li>• enc0: The network model for the enc0 encoder.</li> <li>• hcnn: A hierarchical convolutional neural network.</li> <li>• bilstm: A bidirectional LSTM, in which the signal propagates backward and forward in time. This is an appropriate recurrent neural network (RNN) architecture for sequential learning tasks.</li> <li>• pooled_embedding: The averages of the embeddings of all of the tokens in the input.</li> </ul> <p>Default value: enc0</p>
enc1_cnn_filter_width	<p>The filter width of the CNN enc1 encoder.</p> <p><b>Conditional</b></p> <p>Valid values: <math>1 \leq \text{integer} \leq 9</math></p> <p>Default value: 3</p>
enc1_freeze_pretrained_embedding	<p>Whether to freeze enc1 pretrained embedding weights.</p> <p><b>Conditional</b></p> <p>Valid values: True or False</p> <p>Default value: True</p>

Parameter Name	Description
enc1_layers	<p>The number of layers in the enc1 encoder.</p> <p><b>Conditional</b></p> <p>Valid values: auto or <math>1 \leq \text{integer} \leq 4</math></p> <ul style="list-style-type: none"> <li>For hcnn, auto means 4.</li> <li>For bilstm, auto means 1.</li> <li>For pooled_embedding, auto ignores the number of layers.</li> </ul> <p>Default value: auto</p>
enc1_max_seq_len	<p>The maximum sequence length for the enc1 encoder.</p> <p><b>Conditional</b></p> <p>Valid values: <math>1 \leq \text{integer} \leq 5000</math></p>
enc1_pretrained_embedding_file	<p>The filename of the enc1 pretrained token embedding file in the auxiliary data channel.</p> <p><b>Conditional</b></p> <p>Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9\.\_]</p> <p>Default value: "" (empty string)</p>
enc1_token_embedding_dim	<p>The output dimension of the enc1 token embedding layer.</p> <p><b>Conditional</b></p> <p>Valid values: <math>2 \leq \text{integer} \leq 1000</math></p> <p>Default value: 300</p>
enc1_vocab_file	<p>The vocabulary file for mapping pretrained enc1 token embeddings to vocabulary IDs.</p> <p><b>Conditional</b></p> <p>Valid values: String with alphanumeric characters, underscore, or period. [A-Za-z0-9\.\_]</p> <p>Default value: "" (empty string)</p>
enc1_vocab_size	<p>The vocabulary size of enc0 tokens.</p> <p><b>Conditional</b></p> <p>Valid values: <math>2 \leq \text{integer} \leq 3000000</math></p>

Parameter Name	Description
epochs	<p>The number of epochs to run for training.</p> <p><b>Optional</b></p> <p>Valid values: <math>1 \leq \text{integer} \leq 100</math></p> <p>Default value: 30</p>
learning_rate	<p>The learning rate for training.</p> <p><b>Optional</b></p> <p>Valid values: <math>1.0E-6 \leq \text{float} \leq 1.0</math></p> <p>Default value: 0.0004</p>
mini_batch_size	<p>The batch size that the dataset is split into for an <code>optimizer</code> during training.</p> <p><b>Optional</b></p> <p>Valid values: <math>1 \leq \text{integer} \leq 10000</math></p> <p>Default value: 32</p>
mlp_activation	<p>The type of activation function for the multilayer perceptron (MLP) layer.</p> <p><b>Optional</b></p> <p>Valid values: <code>tanh</code>, <code>relu</code>, or <code>linear</code></p> <ul style="list-style-type: none"> <li>• <code>tanh</code>: Hyperbolic tangent</li> <li>• <code>relu</code>: Rectified linear unit (ReLU)</li> <li>• <code>linear</code>: Linear function</li> </ul> <p>Default value: <code>linear</code></p>
mlp_dim	<p>The dimension of the output from MLP layers.</p> <p><b>Optional</b></p> <p>Valid values: <math>2 \leq \text{integer} \leq 10000</math></p> <p>Default value: 512</p>
mlp_layers	<p>The number of MLP layers in the network.</p> <p><b>Optional</b></p> <p>Valid values: <math>0 \leq \text{integer} \leq 10</math></p> <p>Default value: 2</p>

Parameter Name	Description
<code>negative_sampling_rate</code>	<p>The ratio of negative samples, generated to assist in training the algorithm, to positive samples that are provided by users. Negative samples represent data that is unlikely to occur in reality and are labelled negatively for training. They facilitate training a model to discriminate between the positive samples observed and the negative samples that are not. To specify the ratio of negative samples to positive samples used for training, set the value to a positive integer. For example, if you train the algorithm on input data in which all of the samples are positive and set <code>negative_sampling_rate</code> to 2, the Object2Vec algorithm internally generates two negative samples per positive sample. If you don't want to generate or use negative samples during training, set the value to 0.</p> <p><b>Optional</b></p> <p>Valid values: <math>0 \leq \text{integer}</math></p> <p>Default value: 0 (off)</p>
<code>num_classes</code>	<p>The number of classes for classification training. Amazon SageMaker ignores this hyperparameter for regression problems.</p> <p><b>Optional</b></p> <p>Valid values: <math>2 \leq \text{integer} \leq 30</math></p> <p>Default value: 2</p>
<code>optimizer</code>	<p>The optimizer type.</p> <p><b>Optional</b></p> <p>Valid values: <code>adadelta</code>, <code>adagrad</code>, <code>adam</code>, <code>sgd</code>, or <code>rmsprop</code>.</p> <ul style="list-style-type: none"> <li>• <code>adadelta</code>: A <a href="#">per-dimension learning rate method for gradient descent</a></li> <li>• <code>adagrad</code>: The <a href="#">adaptive gradient algorithm</a></li> <li>• <code>adam</code>: The <a href="#">adaptive moment estimation algorithm</a></li> <li>• <code>sgd</code>: <a href="#">Stochastic gradient descent</a></li> <li>• <code>rmsprop</code>: <a href="#">Root mean square propagation</a></li> </ul> <p>Default value: <code>adam</code></p>

Parameter Name	Description
output_layer	<p>The type of output layer where you specify that the task is regression or classification.</p> <p><b>Optional</b></p> <p>Valid values: <code>softmax</code> or <code>mean_squared_error</code></p> <ul style="list-style-type: none"> <li>• <code>softmax</code>: The <a href="#">Softmax function</a> used for classification.</li> <li>• <code>mean_squared_error</code>: The <a href="#">MSE</a> used for regression.</li> </ul> <p>Default value: <code>softmax</code></p>
tied_token_embedding_weight	<p>Whether to use a shared embedding layer for both encoders. If the inputs to both encoders use the same token-level units, use a shared token embedding layer. For example, for a collection of documents, if one encoder encodes sentences and another encodes whole documents, you can use a shared token embedding layer. That's because both sentences and documents are composed of word tokens from the same vocabulary.</p> <p><b>Optional</b></p> <p>Valid values: <code>True</code> or <code>False</code></p> <p>Default value: <code>False</code></p>
token_embedding_storage_type	<p>The mode of gradient update used during training: when the <code>dense</code> mode is used, the optimizer calculates the full gradient matrix for the token embedding layer even if most rows of the gradient are zero-valued. When <code>sparse</code> mode is used, the optimizer only stores rows of the gradient that are actually being used in the mini-batch. If you want the algorithm to perform lazy gradient updates, which calculate the gradients only in the non-zero rows and which speed up training, specify <code>row_sparse</code>. Setting the value to <code>row_sparse</code> constrains the values available for other hyperparameters, as follows:</p> <ul style="list-style-type: none"> <li>• The optimizer hyperparameter must be set to <code>adam</code>, <code>adagrad</code>, or <code>sgd</code>. Otherwise, the algorithm throws a <code>CustomerValueError</code>.</li> <li>• The algorithm automatically disables bucketing, setting the <code>bucket_width</code> hyperparameter to 0.</li> </ul> <p><b>Optional</b></p> <p>Valid values: <code>dense</code> or <code>row_sparse</code></p> <p>Default value: <code>dense</code></p>

Parameter Name	Description
weight_decay	<p>The weight decay parameter used for optimization.</p> <p><b>Optional</b></p> <p>Valid values: <math>0 \leq \text{float} \leq 10000</math></p> <p>Default value: 0 (no decay)</p>

## Tune an Object2Vec Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. For the objective metric, you use one of the metrics that the algorithm computes. Automatic model tuning searches the chosen hyperparameters to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics Computed by the Object2Vec Algorithm

The Object2Vec algorithm has both classification and regression metrics. The `output_layer` type determines which metric you can use for automatic model tuning.

#### Regressor Metrics Computed by the Object2Vec Algorithm

The algorithm reports a mean squared error regressor metric, which is computed during testing and validation. When tuning the model for regression tasks, choose this metric as the objective.

Metric Name	Description	Optimization Direction
<code>test:mean_squared_error</code>	The Mean Square Error	Minimize
<code>validation:mean_squared_error</code>	The Mean Square Error	Minimize

#### Classification Metrics Computed by the Object2Vec Algorithm

The Object2Vec algorithm reports accuracy and cross-entropy classification metrics, which are computed during test and validation. When tuning the model for classification tasks, choose one of these as the objective.

Metric Name	Description	Optimization Direction
<code>test:accuracy</code>	Accuracy	Maximize
<code>test:cross_entropy</code>	Cross-entropy	Minimize
<code>validation:accuracy</code>	Accuracy	Maximize
<code>validation:cross_entropy</code>	Cross-entropy	Minimize

### Tunable Object2Vec Hyperparameters

You can tune the following hyperparameters for the Object2Vec algorithm.

Hyperparameter Name	Hyperparameter Type	Recommended Ranges and Values	
dropout	ContinuousParameterRange	MinValue: 0.0, MaxValue: 1.0	
early_stopping_patience	IntegerParameterRange	MinValue: 1, MaxValue: 5	
early_stopping_tolerance	ContinuousParameterRange	MinValue: 0.001, MaxValue: 0.1	
enc_dim	IntegerParameterRange	MinValue: 4, MaxValue: 4096	
enc0_cnn_filter_width	IntegerParameterRange	MinValue: 1, MaxValue: 5	
enc0_layers	IntegerParameterRange	MinValue: 1, MaxValue: 4	
enc0_token_embedding_size	IntegerParameterRange	MinValue: 5, MaxValue: 300	
enc1_cnn_filter_width	IntegerParameterRange	MinValue: 1, MaxValue: 5	
enc1_layers	IntegerParameterRange	MinValue: 1, MaxValue: 4	
enc1_token_embedding_size	IntegerParameterRange	MinValue: 5, MaxValue: 300	
epochs	IntegerParameterRange	MinValue: 4, MaxValue: 20	
learning_rate	ContinuousParameterRange	MinValue: 1e-6, MaxValue: 1.0	
mini_batch_size	IntegerParameterRange	MinValue: 1, MaxValue: 8192	
mlp_activation	CategoricalParameterRanges	[tanh, relu, linear]	
mlp_dim	IntegerParameterRange	MinValue: 16, MaxValue: 1024	
mlp_layers	IntegerParameterRange	MinValue: 1, MaxValue: 4	
optimizer	CategoricalParameterRanges	[adagrad, adam, rmsprop, sgd, adadelta]	
weight_decay	ContinuousParameterRange	MinValue: 0.0, MaxValue: 1.0	

## Data Formats for Object2Vec Training

### Input: JSON Lines Request Format

Content-type: application/jsonlines

```
{"label": 0, "in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]}  
{"label": 1, "in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4], "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]}  
{"label": 1, "in0": [774, 14, 21, 206], "in1": [21, 366, 125]}
```

The “in0” and “in1” are the inputs for encoder0 and encoder1, respectively. The same format is valid for both classification and regression problems. For regression, the field “label” can accept real valued inputs.

## Data Formats for Object2Vec Inference

### GPU optimization: Classification or Regression

Due to GPU memory scarcity, the INFERENCER\_PREFERRED\_MODE environment variable can be specified to optimize on whether the classification/regression or the [the section called “Output: Encoder Embeddings” \(p. 1479\)](#) inference network is loaded into GPU. If the majority of your inference is for classification or regression, specify INFERENCER\_PREFERRED\_MODE=classification. The following is a Batch Transform example of using 4 instances of p3.2xlarge that optimizes for classification/regression inference:

```
transformer = o2v.transformer(instance_count=4,  
                             instance_type="ml.p2.xlarge",  
                             max_concurrent_transforms=2,  
                             max_payload=1, # 1MB  
                             strategy='MultiRecord',  
                             env={'INFERENCER_PREFERRED_MODE': 'classification'}, # only  
                             useful with GPU  
                             output_path=output_s3_path)
```

### Input: Classification or Regression Request Format

Content-type: application/json

```
{  
    "instances" : [  
        {"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]},  
        {"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4], "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]},  
        {"in0": [774, 14, 21, 206], "in1": [21, 366, 125]}  
    ]  
}
```

Content-type: application/jsonlines

```
{"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]}  
{"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4], "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]}  
{"in0": [774, 14, 21, 206], "in1": [21, 366, 125]}
```

For classification problems, the length of the scores vector corresponds to `num_classes`. For regression problems, the length is 1.

### Output: Classification or Regression Response Format

Accept: application/json

```
{
    "predictions": [
        {
            "scores": [
                0.6533935070037842,
                0.07582679390907288,
                0.2707797586917877
            ]
        },
        {
            "scores": [
                0.026291321963071823,
                0.6577019095420837,
                0.31600672006607056
            ]
        }
    ]
}
```

Accept: application/jsonlines

```
{"scores": [0.195667684078216, 0.395351558923721, 0.408980727195739]}
{"scores": [0.251988261938095, 0.258233487606048, 0.489778339862823]}
{"scores": [0.280087798833847, 0.368331134319305, 0.351581096649169]}
```

In both the classification and regression formats, the scores apply to individual labels.

### Encoder Embeddings for Object2Vec

#### GPU optimization: Encoder Embeddings

An embedding is a mapping from discrete objects, such as words, to vectors of real numbers.

Due to GPU memory scarcity, the `INFERENCER_PREFERRED_MODE` environment variable can be specified to optimize on whether the [the section called “Inference Formats: Scoring” \(p. 1477\)](#) or the encoder embedding inference network is loaded into GPU. If the majority of your inference is for encoder embeddings, specify `INFERENCER_PREFERRED_MODE=embedding`. The following is a Batch Transform example of using 4 instances of p3.2xlarge that optimizes for encoder embedding inference:

```
transformer = o2v.transformer(instance_count=4,
                             instance_type="ml.p2.xlarge",
                             max_concurrent_transforms=2,
                             max_payload=1, # 1MB
                             strategy='MultiRecord',
                             env={'INFERENCER_PREFERRED_MODE': 'embedding'}, # only useful
                             with GPU
                             output_path=output_s3_path)
```

### Input: Encoder Embeddings

Content-type: application/json; infer\_max\_seqlens=<FWD-LENGTH>,<BCK-LENGTH>

Where <FWD-LENGTH> and <BCK-LENGTH> are integers in the range [1,5000] and define the maximum sequence lengths for the forward and backward encoder.

```
{
  "instances" : [
    {"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4]}, {
      {"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4]}, {
        {"in0": [774, 14, 21, 206]}
    ]
  }
}
```

Content-type: application/jsonlines; infer\_max\_seqlens=<FWD-LENGTH>,<BCK-LENGTH>

Where <FWD-LENGTH> and <BCK-LENGTH> are integers in the range [1,5000] and define the maximum sequence lengths for the forward and backward encoder.

```
{"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4]}
{"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4]}
{"in0": [774, 14, 21, 206]}
```

In both of these formats, you specify only one input type: “in0” or “in1.” The inference service then invokes the corresponding encoder and outputs the embeddings for each of the instances.

### Output: Encoder Embeddings

Content-type: application/json

```
{
  "predictions": [
    {"embeddings":
[0.057368703186511,0.030703511089086,0.099890425801277,0.063688032329082,0.026327300816774,0.0036375711
      {"embeddings":
[0.150190666317939,0.05145975202322,0.098204270005226,0.064249359071254,0.056249320507049,0.01513972133
    ]
  }
}
```

Content-type: application/jsonlines

```
{"embeddings":
[0.057368703186511,0.030703511089086,0.099890425801277,0.063688032329082,0.026327300816774,0.0036375711
{"embeddings":
[0.150190666317939,0.05145975202322,0.098204270005226,0.064249359071254,0.056249320507049,0.01513972133}
```

The vector length of the embeddings output by the inference service is equal to the value of one of the following hyperparameters that you specify at training time: enc0\_token\_embedding\_dim, enc1\_token\_embedding\_dim, or enc\_dim.

## Object Detection Algorithm

The Amazon SageMaker Object Detection algorithm detects and classifies objects in images using a single deep neural network. It is a supervised learning algorithm that takes images as input and identifies all instances of objects within the image scene. The object is categorized into one of the classes in a specified collection with a confidence score that it belongs to the class. Its location and scale in the image are indicated by a rectangular bounding box. It uses the [Single Shot multibox Detector \(SSD\)](#) framework and supports two base networks: [VGG](#) and [ResNet](#). The network can be trained from scratch, or trained with models that have been pre-trained on the [ImageNet](#) dataset.

### Topics

- [Input/Output Interface for the Object Detection Algorithm \(p. 1480\)](#)
- [EC2 Instance Recommendation for the Object Detection Algorithm \(p. 1482\)](#)

- [Object Detection Sample Notebooks \(p. 1482\)](#)
- [How Object Detection Works \(p. 1483\)](#)
- [Object Detection Hyperparameters \(p. 1483\)](#)
- [Tune an Object Detection Model \(p. 1488\)](#)
- [Object Detection Request and Response Formats \(p. 1489\)](#)

## Input/Output Interface for the Object Detection Algorithm

The SageMaker Object Detection algorithm supports both RecordIO (`application/x-recordio`) and image (`image/png`, `image/jpeg`, and `application/x-image`) content types for training in file mode and supports RecordIO (`application/x-recordio`) for training in pipe mode. However you can also train in pipe mode using the image files (`image/png`, `image/jpeg`, and `application/x-image`), without creating RecordIO files, by using the augmented manifest format. The recommended input format for the Amazon SageMaker object detection algorithms is [Apache MXNet RecordIO](#). However, you can also use raw images in `.jpg` or `.png` format. The algorithm supports only `application/x-image` for inference.

### Note

To maintain better interoperability with existing deep learning frameworks, this differs from the protobuf data formats commonly used by other Amazon SageMaker algorithms.

See the [Object Detection Sample Notebooks \(p. 1482\)](#) for more details on data formats.

### Train with the RecordIO Format

If you use the RecordIO format for training, specify both train and validation channels as values for the `InputDataConfig` parameter of the [CreateTrainingJob](#) request. Specify one RecordIO (.rec) file in the train channel and one RecordIO file in the validation channel. Set the content type for both channels to `application/x-recordio`. An example of how to generate RecordIO file can be found in the object detection sample notebook. You can also use tools from the [MXNet's GluonCV](#) to generate RecordIO files for popular datasets like the [PASCAL Visual Object Classes](#) and [Common Objects in Context \(COCO\)](#).

### Train with the Image Format

If you use the image format for training, specify `train`, `validation`, `train_annotation`, and `validation_annotation` channels as values for the `InputDataConfig` parameter of [CreateTrainingJob](#) request. Specify the individual image data (.jpg or .png) files for the train and validation channels. For annotation data, you can use the JSON format. Specify the corresponding .json files in the `train_annotation` and `validation_annotation` channels. Set the content type for all four channels to `image/png` or `image/jpeg` based on the image type. You can also use the content type `application/x-image` when your dataset contains both .jpg and .png images. The following is an example of a .json file.

```
{  
    "file": "your_image_directory/sample_image1.jpg",  
    "image_size": [  
        {  
            "width": 500,  
            "height": 400,  
            "depth": 3  
        }  
    ],  
    "annotations": [  
        {  
            "class_id": 0,  
            "left": 111,  
            "top": 134,  
            "width": 61,  
            "height": 128  
        }  
    ]  
}
```

```

},
{
    "class_id": 0,
    "left": 161,
    "top": 250,
    "width": 79,
    "height": 143
},
{
    "class_id": 1,
    "left": 101,
    "top": 185,
    "width": 42,
    "height": 130
}
],
"categories": [
{
    "class_id": 0,
    "name": "dog"
},
{
    "class_id": 1,
    "name": "cat"
}
]
}

```

Each image needs a .json file for annotation, and the .json file should have the same name as the corresponding image. The name of above .json file should be "sample\_image1.json". There are four properties in the annotation .json file. The property "file" specifies the relative path of the image file. For example, if your training images and corresponding .json files are stored in s3://*your\_bucket*/train/sample\_image and s3://*your\_bucket*/train\_annotation, specify the path for your train and train\_annotation channels as s3://*your\_bucket*/train and s3://*your\_bucket*/train\_annotation, respectively.

In the .json file, the relative path for an image named sample\_image1.jpg should be sample\_image/sample\_image1.jpg. The "image\_size" property specifies the overall image dimensions. The SageMaker object detection algorithm currently only supports 3-channel images. The "annotations" property specifies the categories and bounding boxes for objects within the image. Each object is annotated by a "class\_id" index and by four bounding box coordinates ("left", "top", "width", "height"). The "left" (x-coordinate) and "top" (y-coordinate) values represent the upper-left corner of the bounding box. The "width" (x-coordinate) and "height" (y-coordinate) values represent the dimensions of the bounding box. The origin (0, 0) is the upper-left corner of the entire image. If you have multiple objects within one image, all the annotations should be included in a single .json file. The "categories" property stores the mapping between the class index and class name. The class indices should be numbered successively and the numbering should start with 0. The "categories" property is optional for the annotation .json file

### Train with Augmented Manifest Image Format

The augmented manifest format enables you to do training in pipe mode using image files without needing to create RecordIO files. You need to specify both train and validation channels as values for the `InputDataConfig` parameter of the `CreateTrainingJob` request. While using the format, an S3 manifest file needs to be generated that contains the list of images and their corresponding annotations. The manifest file format should be in `JSON Lines` format in which each line represents one sample. The images are specified using the '`source-ref`' tag that points to the S3 location of the image. The annotations are provided under the "`AttributeNames`" parameter value as specified in the `CreateTrainingJob` request. It can also contain additional metadata under the `metadata` tag, but these are ignored by the algorithm. In the following example, the "`AttributeNames`" are contained in the list [`"source-ref"`, `"bounding-box"`]:

```
{"source-ref": "s3://your_bucket/image1.jpg", "bounding-box": {"image_size": [{"width": 500, "height": 400, "depth": 3}], "annotations": [{"class_id": 0, "left": 111, "top": 134, "width": 61, "height": 128}, {"class_id": 5, "left": 161, "top": 250, "width": 80, "height": 50}]}, "bounding-box-metadata": {"class-map": {"0": "dog", "5": "horse"}, "type": "groundtruth/object-detection"}}, {"source-ref": "s3://your_bucket/image2.jpg", "bounding-box": {"image_size": [{"width": 400, "height": 300, "depth": 3}], "annotations": [{"class_id": 1, "left": 100, "top": 120, "width": 43, "height": 78}]}, "bounding-box-metadata": {"class-map": {"1": "cat"}, "type": "groundtruth/object-detection"}}}
```

The order of "AttributeNames" in the input files matters when training the Object Detection algorithm. It accepts piped data in a specific order, with `image` first, followed by `annotations`. So the "AttributeNames" in this example are provided with "source-ref" first, followed by "bounding-box". When using Object Detection with Augmented Manifest, the value of parameter `RecordWrapperType` must be set as "RecordIO".

For more information on augmented manifest files, see [Provide Dataset Metadata to Training Jobs with an Augmented Manifest File \(p. 1864\)](#).

### Incremental Training

You can also seed the training of a new model with the artifacts from a model that you trained previously with SageMaker. Incremental training saves training time when you want to train a new model with the same or similar data. SageMaker object detection models can be seeded only with another built-in object detection model trained in SageMaker.

To use a pretrained model, in the `CreateTrainingJob` request, specify the `ChannelName` as "model" in the `InputDataConfig` parameter. Set the `ContentType` for the model channel to `application/x-sagemaker-model`. The input hyperparameters of both the new model and the pretrained model that you upload to the model channel must have the same settings for the `base_network` and `num_classes` input parameters. These parameters define the network architecture. For the pretrained model file, use the compressed model artifacts (in `.tar.gz` format) output by SageMaker. You can use either RecordIO or image formats for input data.

For a sample notebook that shows how to use incremental training with the SageMaker object detection algorithm, see [SageMaker Object Detection Incremental Training](#) sample notebook. For more information on incremental training and for instructions on how to use it, see [Incremental Training in Amazon SageMaker \(p. 1855\)](#).

### EC2 Instance Recommendation for the Object Detection Algorithm

For object detection, we support the following GPU instances for training: `ml.p2.xlarge`, `ml.p2.8xlarge`, `ml.p2.16xlarge`, `ml.p3.2xlarge`, `ml.p3.8xlarge` and `ml.p3.16xlarge`. We recommend using GPU instances with more memory for training with large batch sizes. You can also run the algorithm on multi-GPU and multi-machine settings for distributed training. However, both CPU (such as C5 and M5) and GPU (such as P2 and P3) instances can be used for the inference. All the supported instance types for inference are itemized on [Amazon SageMaker ML Instance Types](#).

### Object Detection Sample Notebooks

For a sample notebook that shows how to use the SageMaker Object Detection algorithm to train and host a model on the COCO dataset using the Single Shot multibox Detector algorithm, see [Object Detection using the Image and JSON format](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). Once you have created a notebook instance and opened it, select the `SageMaker Examples` tab to see a list of all the SageMaker samples. The object detection example notebook using the Object Detection algorithm is located in the `Introduction to Amazon Algorithms` section. To open a notebook, click on its `Use` tab and select `Create copy`.

## How Object Detection Works

The object detection algorithm identifies and locates all instances of objects in an image from a known collection of object categories. The algorithm takes an image as input and outputs the category that the object belongs to, along with a confidence score that it belongs to the category. The algorithm also predicts the object's location and scale with a rectangular bounding box. Amazon SageMaker Object Detection uses the [Single Shot multibox Detector \(SSD\)](#) algorithm that takes a convolutional neural network (CNN) pretrained for classification task as the base network. SSD uses the output of intermediate layers as features for detection.

Various CNNs such as [VGG](#) and [ResNet](#) have achieved great performance on the image classification task. Object detection in Amazon SageMaker supports both VGG-16 and ResNet-50 as a base network for SSD. The algorithm can be trained in full training mode or in transfer learning mode. In full training mode, the base network is initialized with random weights and then trained on user data. In transfer learning mode, the base network weights are loaded from pretrained models.

The object detection algorithm uses standard data augmentation operations, such as flip, rescale, and jitter, on the fly internally to help avoid overfitting.

## Object Detection Hyperparameters

In the [CreateTrainingJob](#) request, you specify the training algorithm that you want to use. You can also specify algorithm-specific hyperparameters that are used to help estimate the parameters of the model from a training dataset. The following table lists the hyperparameters provided by Amazon SageMaker for training the object detection algorithm. For more information about how object training works, see [How Object Detection Works \(p. 1483\)](#).

Parameter Name	Description
<code>num_classes</code>	<p>The number of output classes. This parameter defines the dimensions of the network output and is typically set to the number of classes in the dataset.</p> <p><b>Required</b></p> <p>Valid values: positive integer</p>
<code>num_training_samples</code>	<p>The number of training examples in the input dataset.</p> <p><b>Note</b> If there is a mismatch between this value and the number of samples in the training set, then the behavior of the <code>lr_scheduler_step</code> parameter will be undefined and distributed training accuracy may be affected.</p> <p><b>Required</b></p> <p>Valid values: positive integer</p>
<code>base_network</code>	<p>The base network architecture to use.</p> <p><b>Optional</b></p> <p>Valid values: 'vgg-16' or 'resnet-50'</p> <p>Default value: 'vgg-16'</p>
<code>early_stopping</code>	True to use early stopping logic during training. False not to use it.

Parameter Name	Description
	<b>Optional</b> Valid values: <code>True</code> or <code>False</code> Default value: <code>False</code>
<code>early_stopping_min_epochs</code>	The minimum number of epochs that must be run before the early stopping logic can be invoked. It is used only when <code>early_stopping = True</code> . <b>Optional</b> Valid values: positive integer Default value: 10
<code>early_stopping_patience</code>	The number of epochs to wait before ending training if no improvement, as defined by the <code>early_stopping_tolerance</code> hyperparameter, is made in the relevant metric. It is used only when <code>early_stopping = True</code> . <b>Optional</b> Valid values: positive integer Default value: 5
<code>early_stopping_tolerance</code>	The tolerance value that the relative improvement in <code>validation:mAP</code> , the mean average precision (mAP), is required to exceed to avoid early stopping. If the ratio of the change in the mAP divided by the previous best mAP is smaller than the <code>early_stopping_tolerance</code> value set, early stopping considers that there is no improvement. It is used only when <code>early_stopping = True</code> . <b>Optional</b> Valid values: $0 \leq \text{float} \leq 1$ Default value: 0.0
<code>image_shape</code>	The image size for input images. We rescale the input image to a square image with this size. We recommend using 300 and 512 for better performance. <b>Optional</b> Valid values: positive integer $\geq 300$ Default: 300
<code>epochs</code>	The number of training epochs. <b>Optional</b> Valid values: positive integer Default: 30

Parameter Name	Description
<code>freeze_layer_pattern</code>	<p>The regular expression (regex) for freezing layers in the base network. For example, if we set <code>freeze_layer_pattern = "^(conv1_ conv2_).*</code>", then any layers with a name that contains "conv1_" or "conv2_" are frozen, which means that the weights for these layers are not updated during training. The layer names can be found in the network symbol files <a href="#">vgg16-symbol.json</a> and <a href="#">resnet-50-symbol.json</a>. Freezing a layer means that its weights can not be modified further. This can reduce training time significantly in exchange for modest losses in accuracy. This technique is commonly used in transfer learning where the lower layers in the base network do not need to be retrained.</p> <p><b>Optional</b></p> <p>Valid values: string</p> <p>Default: No layers frozen.</p>
<code>kv_store</code>	<p>The weight update synchronization mode used for distributed training. The weights can be updated either synchronously or asynchronously across machines. Synchronous updates typically provide better accuracy than asynchronous updates but can be slower. See the <a href="#">Distributed Training</a> MXNet tutorial for details.</p> <p><b>Note</b>  This parameter is not applicable to single machine training.</p> <p><b>Optional</b></p> <p>Valid values: 'dist_sync' or 'dist_async'</p> <ul style="list-style-type: none"> <li>'dist_sync': The gradients are synchronized after every batch with all the workers. With 'dist_sync', batch-size now means the batch size used on each machine. So if there are n machines and we use batch size b, then dist_sync behaves like a single machine with batch size n*b.</li> <li>'dist_async': Performs asynchronous updates. The weights are updated whenever gradients are received from any machine and the weight updates are atomic. However, the order is not guaranteed.</li> </ul> <p>Default: -</p>

Parameter Name	Description
<code>label_width</code>	<p>The force padding label width used to sync across training and validation data. For example, if one image in the data contains at most 10 objects, and each object's annotation is specified with 5 numbers, [class_id, left, top, width, height], then the <code>label_width</code> should be no smaller than <math>(10*5 + \text{header information length})</math>. The header information length is usually 2. We recommend using a slightly larger <code>label_width</code> for the training, such as 60 for this example.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer large enough to accommodate the largest annotation information length in the data.</p> <p>Default: 350</p>
<code>learning_rate</code>	<p>The initial learning rate.</p> <p><b>Optional</b></p> <p>Valid values: float in (0, 1]</p> <p>Default: 0.001</p>
<code>lr_scheduler_factor</code>	<p>The ratio to reduce learning rate. Used in conjunction with the <code>lr_scheduler_step</code> parameter defined as <math>\text{lr\_new} = \text{lr\_old} * \text{lr\_scheduler\_factor}</math>.</p> <p><b>Optional</b></p> <p>Valid values: float in (0, 1)</p> <p>Default: 0.1</p>
<code>lr_scheduler_step</code>	<p>The epochs at which to reduce the learning rate. The learning rate is reduced by <code>lr_scheduler_factor</code> at epochs listed in a comma-delimited string: "epoch1, epoch2, ...". For example, if the value is set to "10, 20" and the <code>lr_scheduler_factor</code> is set to 1/2, then the learning rate is halved after 10th epoch and then halved again after 20th epoch.</p> <p><b>Optional</b></p> <p>Valid values: string</p> <p>Default: empty string</p>

Parameter Name	Description
mini_batch_size	<p>The batch size for training. In a single-machine multi-gpu setting, each GPU handles <code>mini_batch_size/num_gpu</code> training samples. For the multi-machine training in <code>dist_sync</code> mode, the actual batch size is <code>mini_batch_size*number of machines</code>. A large <code>mini_batch_size</code> usually leads to faster training, but it may cause out of memory problem. The memory usage is related to <code>mini_batch_size</code>, <code>image_shape</code>, and <code>base_network</code> architecture. For example, on a single p3.2xlarge instance, the largest <code>mini_batch_size</code> without an out of memory error is 32 with the <code>base_network</code> set to "resnet-50" and an <code>image_shape</code> of 300. With the same instance, you can use 64 as the <code>mini_batch_size</code> with the base network vgg-16 and an <code>image_shape</code> of 300.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default: 32</p>
momentum	<p>The momentum for sgd. Ignored for other optimizers.</p> <p><b>Optional</b></p> <p>Valid values: float in (0, 1]</p> <p>Default: 0.9</p>
nms_threshold	<p>The non-maximum suppression threshold.</p> <p><b>Optional</b></p> <p>Valid values: float in (0, 1]</p> <p>Default: 0.45</p>
optimizer	<p>The optimizer types. For details on optimizer values, see <a href="#">MXNet's API</a>.</p> <p><b>Optional</b></p> <p>Valid values: ['sgd', 'adam', 'rmsprop', 'adadelta']</p> <p>Default: 'sgd'</p>
overlap_threshold	<p>The evaluation overlap threshold.</p> <p><b>Optional</b></p> <p>Valid values: float in (0, 1]</p> <p>Default: 0.5</p>

Parameter Name	Description
use_pretrained_model	<p>Indicates whether to use a pre-trained model for training. If set to 1, then the pre-trained model with corresponding architecture is loaded and used for training. Otherwise, the network is trained from scratch.</p> <p><b>Optional</b></p> <p>Valid values: 0 or 1</p> <p>Default: 1</p>
weight_decay	<p>The weight decay coefficient for sgd and rmsprop. Ignored for other optimizers.</p> <p><b>Optional</b></p> <p>Valid values: float in (0, 1)</p> <p>Default: 0.0005</p>

## Tune an Object Detection Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics Computed by the Object Detection Algorithm

The object detection algorithm reports on a single metric during training: validation:mAP. When tuning a model, choose this metric as the objective metric.

Metric Name	Description	Optimization Direction
validation:mAP	Mean Average Precision (mAP) computed on the validation set.	Maximize

### Tunable Object Detection Hyperparameters

Tune the Amazon SageMaker object detection model with the following hyperparameters. The hyperparameters that have the greatest impact on the object detection objective metric are: mini\_batch\_size, learning\_rate, and optimizer.

Parameter Name	Parameter Type	Recommended Ranges
learning_rate	ContinuousParameterRange	MinValue: 1e-6, MaxValue: 0.5
mini_batch_size	IntegerParameterRanges	MinValue: 8, MaxValue: 64

Parameter Name	Parameter Type	Recommended Ranges
momentum	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.999
optimizer	CategoricalParameterRanges	['sgd', 'adam', 'rmsprop', 'adadelta']
weight_decay	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.999

## Object Detection Request and Response Formats

### Request Format

Query a trained model by using the model's endpoint. The endpoint takes .jpg and .png image formats with `image/jpeg` and `image/png` content-types.

### Response Formats

The response is the class index with a confidence score and bounding box coordinates for all objects within the image encoded in JSON format. The following is an example of response .json file:

```
{"prediction": [
    [4.0, 0.86419455409049988, 0.3088374733924866, 0.07030484080314636, 0.7110607028007507,
     0.9345266819000244],
    [0.0, 0.73376623392105103, 0.5714187026023865, 0.40427327156066895, 0.827075183391571,
     0.9712159633636475],
    [4.0, 0.32643985450267792, 0.3677481412887573, 0.034883320331573486, 0.6318609714508057,
     0.5967587828636169],
    [8.0, 0.22552496790885925, 0.6152569651603699, 0.5722782611846924, 0.882301390171051,
     0.8985623121261597],
    [3.0, 0.42260299175977707, 0.019305512309074402, 0.08386176824569702,
     0.39093565940856934, 0.9574796557426453]
]}
```

Each row in this .json file contains an array that represents a detected object. Each of these object arrays consists of a list of six numbers. The first number is the predicted class label. The second number is the associated confidence score for the detection. The last four numbers represent the bounding box coordinates [xmin, ymin, xmax, ymax]. These output bounding box corner indices are normalized by the overall image size. Note that this encoding is different than that use by the input .json format. For example, in the first entry of the detection result, 0.3088374733924866 is the left coordinate (x-coordinate of upper-left corner) of the bounding box as a ratio of the overall image width, 0.07030484080314636 is the top coordinate (y-coordinate of upper-left corner) of the bounding box as a ratio of the overall image height, 0.7110607028007507 is the right coordinate (x-coordinate of lower-right corner) of the bounding box as a ratio of the overall image width, and 0.9345266819000244 is the bottom coordinate (y-coordinate of lower-right corner) of the bounding box as a ratio of the overall image height.

To avoid unreliable detection results, you might want to filter out the detection results with low confidence scores. In the [object detection sample notebook](#), we provide scripts to remove the low confidence detections. Scripts are also provided to plot the bounding boxes on the original image.

For batch transform, the response is in JSON format, where the format is identical to the JSON format described above. The detection results of each image is represented as a JSON file. For example:

```
{"prediction": [[label_id, confidence_score, xmin, ymin, xmax, ymax], [label_id,
confidence_score, xmin, ymin, xmax, ymax]]}
```

For more details on training and inference, see the [Object Detection Sample Notebooks \(p. 1482\)](#).

### OUTPUT: JSON Response Format

accept: application/json;annotation=1

```
{  
    "image_size": [  
        {  
            "width": 500,  
            "height": 400,  
            "depth": 3  
        }  
    ],  
    "annotations": [  
        {  
            "class_id": 0,  
            "score": 0.943,  
            "left": 111,  
            "top": 134,  
            "width": 61,  
            "height": 128  
        },  
        {  
            "class_id": 0,  
            "score": 0.0013,  
            "left": 161,  
            "top": 250,  
            "width": 79,  
            "height": 143  
        },  
        {  
            "class_id": 1,  
            "score": 0.0133,  
            "left": 101,  
            "top": 185,  
            "width": 42,  
            "height": 130  
        }  
    ]  
}
```

## Principal Component Analysis (PCA) Algorithm

PCA is an unsupervised machine learning algorithm that attempts to reduce the dimensionality (number of features) within a dataset while still retaining as much information as possible. This is done by finding a new set of features called *components*, which are composites of the original features that are uncorrelated with one another. They are also constrained so that the first component accounts for the largest possible variability in the data, the second component the second most variability, and so on.

In Amazon SageMaker, PCA operates in two modes, depending on the scenario:

- **regular:** For datasets with sparse data and a moderate number of observations and features.
- **randomized:** For datasets with both a large number of observations and features. This mode uses an approximation algorithm.

PCA uses tabular data.

The rows represent observations you want to embed in a lower dimensional space. The columns represent features that you want to find a reduced approximation for. The algorithm calculates the

covariance matrix (or an approximation thereof in a distributed manner), and then performs the singular value decomposition on this summary to produce the principal components.

### Topics

- [Input/Output Interface for the PCA Algorithm \(p. 1491\)](#)
- [EC2 Instance Recommendation for the PCA Algorithm \(p. 1491\)](#)
- [PCA Sample Notebooks \(p. 1491\)](#)
- [How PCA Works \(p. 1491\)](#)
- [PCA Hyperparameters \(p. 1492\)](#)
- [PCA Response Formats \(p. 1493\)](#)

## Input/Output Interface for the PCA Algorithm

For training, PCA expects data provided in the train channel, and optionally supports a dataset passed to the test dataset, which is scored by the final algorithm. Both `recordIO-wrapped-protobuf` and `CSV` formats are supported for training. You can use either File mode or Pipe mode to train models on data that is formatted as `recordIO-wrapped-protobuf` or as `CSV`.

For inference, PCA supports `text/csv`, `application/json`, and `application/x-recordio-protobuf`. Results are returned in either `application/json` or `application/x-recordio-protobuf` format with a vector of "projections."

For more information on input and output file formats, see [PCA Response Formats \(p. 1493\)](#) for inference and the [PCA Sample Notebooks \(p. 1491\)](#).

## EC2 Instance Recommendation for the PCA Algorithm

PCA supports both GPU and CPU computation. Which instance type is most performant depends heavily on the specifics of the input data.

## PCA Sample Notebooks

For a sample notebook that shows how to use the SageMaker Principal Component Analysis algorithm to analyze the images of handwritten digits from zero to nine in the MNIST dataset, see [An Introduction to PCA with MNIST](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

## How PCA Works

Principal Component Analysis (PCA) is a learning algorithm that reduces the dimensionality (number of features) within a dataset while still retaining as much information as possible.

PCA reduces dimensionality by finding a new set of features called *components*, which are composites of the original features, but are uncorrelated with one another. The first component accounts for the largest possible variability in the data, the second component the second most variability, and so on.

It is an unsupervised dimensionality reduction algorithm. In unsupervised learning, labels that might be associated with the objects in the training dataset aren't used.

Given the input of a matrix with rows  $\mathbb{F}_1, \dots, \mathbb{F}_n$  each of dimension  $1 \times d$ , the data is partitioned into mini-batches of rows and distributed among the training nodes (workers). Each worker then computes a summary of its data. The summaries of the different workers are then unified into a single solution at the end of the computation.

## Modes

The Amazon SageMaker PCA algorithm uses either of two modes to calculate these summaries, depending on the situation:

- **regular**: for datasets with sparse data and a moderate number of observations and features.
- **randomized**: for datasets with both a large number of observations and features. This mode uses an approximation algorithm.

As the algorithm's last step, it performs the singular value decomposition on the unified solution, from which the principal components are then derived.

### Mode 1: Regular

The workers jointly compute both  $\sum x_i^T x_i$  and  $\sum x_i$ .

#### Note

Because  $x_i$  are  $1 \times d$  row vectors,  $x_i^T x_i$  is a matrix (not a scalar). Using row vectors within the code allows us to obtain efficient caching.

The covariance matrix is computed as  $\sum x_i^T x_i - (1/n)(\sum x_i)^T \sum x_i$ , and its top `num_components` singular vectors form the model.

#### Note

If `subtract_mean` is `False`, we avoid computing and subtracting  $\sum x_i$ .

Use this algorithm when the dimension  $d$  of the vectors is small enough so that  $x_i$  can fit in memory.

### Mode 2: Randomized

When the number of features in the input dataset is large, we use a method to approximate the covariance metric. For every mini-batch  $X_i$  of dimension  $b \times d$ , we randomly initialize a  $(\text{num\_components} + \text{extra\_components}) \times b$  matrix that we multiply by each mini-batch, to create a  $(\text{num\_components} + \text{extra\_components}) \times d$  matrix. The sum of these matrices is computed by the workers, and the servers perform SVD on the final  $(\text{num\_components} + \text{extra\_components}) \times d$  matrix. The top right `num_components` singular vectors of it are the approximation of the top singular vectors of the input matrix.

Let  $t = \text{num\_components} + \text{extra\_components}$ . Given a mini-batch  $X_i$  of dimension  $b \times d$ , the worker draws a random matrix  $H_i$  of dimension  $t \times b$ . Depending on whether the environment uses a GPU or CPU and the dimension size, the matrix is either a random sign matrix where each entry is  $+/-1$  or a *FJLT* (fast Johnson Lindenstrauss transform; for information, see [FJLT Transforms](#) and the follow-up papers). The worker then computes  $H_i X_i$  and maintains  $B = \sum H_i X_i$ . The worker also maintains  $H^T$ , the sum of columns of  $H_1, \dots, H_T$  ( $T$  being the total number of mini-batches), and  $s$ , the sum of all input rows. After processing the entire shard of data, the worker sends the server  $B$ ,  $h$ ,  $s$ , and  $n$  (the number of input rows).

Denote the different inputs to the server as  $B, h, s, n$ . The server computes  $B, h, s, n$  the sums of the respective inputs. It then computes  $C = B - (1/n)h^T h$ , and finds its singular value decomposition. The top-right singular vectors and singular values of  $C$  are used as the approximate solution to the problem.

## PCA Hyperparameters

In the `CreateTrainingJob` request, you specify the training algorithm. You can also specify algorithm-specific HyperParameters as string-to-string maps. The following table lists the hyperparameters for the PCA training algorithm provided by Amazon SageMaker. For more information about how PCA works, see [How PCA Works \(p. 1491\)](#).

Parameter Name	Description
<code>feature_dim</code>	<p>Input dimension.</p> <p><b>Required</b></p> <p>Valid values: positive integer</p>
<code>mini_batch_size</code>	<p>Number of rows in a mini-batch.</p> <p><b>Required</b></p> <p>Valid values: positive integer</p>
<code>num_components</code>	<p>The number of principal components to compute.</p> <p><b>Required</b></p> <p>Valid values: positive integer</p>
<code>algorithm_mode</code>	<p>Mode for computing the principal components.</p> <p><b>Optional</b></p> <p>Valid values: <i>regular</i> or <i>randomized</i></p> <p>Default value: <i>regular</i></p>
<code>extra_components</code>	<p>As the value increases, the solution becomes more accurate but the runtime and memory consumption increase linearly. The default, -1, means the maximum of 10 and <code>num_components</code>. Valid for <i>randomized</i> mode only.</p> <p><b>Optional</b></p> <p>Valid values: Non-negative integer or -1</p> <p>Default value: -1</p>
<code>subtract_mean</code>	<p>Indicates whether the data should be unbiased both during training and at inference.</p> <p><b>Optional</b></p> <p>Valid values: One of <i>true</i> or <i>false</i></p> <p>Default value: <i>true</i></p>

## PCA Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats - Inference](#). This topic contains a list of the available output formats for the SageMaker PCA algorithm.

### JSON Response Format

Accept—application/json

{

```

    "projections": [
        {
            "projection": [1.0, 2.0, 3.0, 4.0, 5.0]
        },
        {
            "projection": [6.0, 7.0, 8.0, 9.0, 0.0]
        },
        ....
    ]
}

```

### JSONLINES Response Format

Accept—application/jsonlines

```

{ "projection": [1.0, 2.0, 3.0, 4.0, 5.0] }
{ "projection": [6.0, 7.0, 8.0, 9.0, 0.0] }

```

### RECORDIO Response Format

Accept—application/x-recordio-protobuf

```

[
    Record = {
        features = {},
        label = {
            'projection': {
                keys: [],
                values: [1.0, 2.0, 3.0, 4.0, 5.0]
            }
        }
    },
    Record = {
        features = {},
        label = {
            'projection': {
                keys: [],
                values: [1.0, 2.0, 3.0, 4.0, 5.0]
            }
        }
    }
]

```

## Random Cut Forest (RCF) Algorithm

Amazon SageMaker Random Cut Forest (RCF) is an unsupervised algorithm for detecting anomalous data points within a data set. These are observations which diverge from otherwise well-structured or patterned data. Anomalies can manifest as unexpected spikes in time series data, breaks in periodicity, or unclassifiable data points. They are easy to describe in that, when viewed in a plot, they are often easily distinguishable from the "regular" data. Including these anomalies in a data set can drastically increase the complexity of a machine learning task since the "regular" data can often be described with a simple model.

With each data point, RCF associates an anomaly score. Low score values indicate that the data point is considered "normal." High values indicate the presence of an anomaly in the data. The definitions of "low" and "high" depend on the application but common practice suggests that scores beyond three standard deviations from the mean score are considered anomalous.

While there are many applications of anomaly detection algorithms to one-dimensional time series data such as traffic volume analysis or sound volume spike detection, RCF is designed to work with arbitrary-

dimensional input. Amazon SageMaker RCF scales well with respect to number of features, data set size, and number of instances.

### Topics

- [Input/Output Interface for the RCF Algorithm \(p. 1495\)](#)
- [Instance Recommendations for the RCF Algorithm \(p. 1496\)](#)
- [RCF Sample Notebooks \(p. 1496\)](#)
- [How RCF Works \(p. 1496\)](#)
- [RCF Hyperparameters \(p. 1499\)](#)
- [Tune an RCF Model \(p. 1499\)](#)
- [RCF Response Formats \(p. 1500\)](#)

## Input/Output Interface for the RCF Algorithm

Amazon SageMaker Random Cut Forest supports the `train` and `test` data channels. The optional `test` channel is used to compute accuracy, precision, recall, and F1-score metrics on labeled data. Train and test data content types can be either `application/x-recordio-protobuf` or `text/csv` formats. For the test data, when using `text/csv` format, the content must be specified as `text/csv;label_size=1` where the first column of each row represents the anomaly label: "1" for an anomalous data point and "0" for a normal data point. You can use either File mode or Pipe mode to train RCF models on data that is formatted as `recordIO-wrapped-protobuf` or as CSV

Also note that the `train` channel only supports `S3DataDistributionType=ShardedByS3Key` and the `test` channel only supports `S3DataDistributionType=FullyReplicated`. The S3 distribution type can be specified using the [Amazon SageMaker Python SDK](#) as follows:

```
import sagemaker

# specify Random Cut Forest training job information and hyperparameters
rcf = sagemaker.estimator.Estimator(...)

# explicitly specify "ShardedByS3Key" distribution type
train_data = sagemaker.inputs.s3_input(
    s3_data=s3_training_data_location,
    content_type='text/csv;label_size=0',
    distribution='ShardedByS3Key')

# run the training job on input data stored in S3
rcf.fit({'train': train_data})
```

To avoid common errors around execution roles, ensure that you have the execution roles required, `AmazonSageMakerFullAccess` and `AmazonEC2ContainerRegistryFullAccess`. To avoid common errors around your image not existing or its permissions being incorrect, ensure that your ECR image is not larger than the allocated disk space on the training instance. To avoid this, run your training job on an instance that has sufficient disk space. In addition, if your ECR image is from a different Amazon account's Elastic Container Service (ECS) repository, and you do not set repository permissions to grant access, this will result in an error. See the [ECR repository permissions](#) for more information on setting a repository policy statement.

See the [S3DataSource](#) for more information on customizing the S3 data source attributes. Finally, in order to take advantage of multi-instance training the training data must be partitioned into at least as many files as instances.

For inference, RCF supports `application/x-recordio-protobuf`, `text/csv` and `application/json` input data content types. See the [Common Data Formats for Built-in Algorithms \(p. 1354\)](#)

documentation for more information. RCF inference returns application/x-recordio-protobuf or application/json formatted output. Each record in these output data contains the corresponding anomaly scores for each input data point. See [Common Data Formats--Inference](#) for more information.

For more information on input and output file formats, see [RCF Response Formats \(p. 1500\)](#) for inference and the [RCF Sample Notebooks \(p. 1496\)](#).

## Instance Recommendations for the RCF Algorithm

For training, we recommend the m1.m4, m1.c4, and m1.c5 instance families. For inference we recommend using a m1.c5.xl instance type in particular, for maximum performance as well as minimized cost per hour of usage. Although the algorithm could technically run on GPU instance types it does not take advantage of GPU hardware.

## RCF Sample Notebooks

For an example of how to train an RCF model and perform inferences with it, see the [An Introduction to SageMaker Random Cut Forests](#) notebook. For instructions how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the SageMaker samples. To open a notebook, click on its **Use** tab and select **Create copy**.

## How RCF Works

Amazon SageMaker Random Cut Forest (RCF) is an unsupervised algorithm for detecting anomalous data points within a dataset. These are observations which diverge from otherwise well-structured or patterned data. Anomalies can manifest as unexpected spikes in time series data, breaks in periodicity, or unclassifiable data points. They are easy to describe in that, when viewed in a plot, they are often easily distinguishable from the "regular" data. Including these anomalies in a dataset can drastically increase the complexity of a machine learning task since the "regular" data can often be described with a simple model.

The main idea behind the RCF algorithm is to create a forest of trees where each tree is obtained using a partition of a sample of the training data. For example, a random sample of the input data is first determined. The random sample is then partitioned according to the number of trees in the forest. Each tree is given such a partition and organizes that subset of points into a k-d tree. The anomaly score assigned to a data point by the tree is defined as the expected change in complexity of the tree as a result adding that point to the tree; which, in approximation, is inversely proportional to the resulting depth of the point in the tree. The random cut forest assigns an anomaly score by computing the average score from each constituent tree and scaling the result with respect to the sample size. The RCF algorithm is based on the one described in reference [1].

## Sample Data Randomly

The first step in the RCF algorithm is to obtain a random sample of the training data. In particular, suppose we want a sample of size  $K$  from  $N$  total data points. If the training data is small enough, the entire dataset can be used, and we could randomly draw  $K$  elements from this set. However, frequently the training data is too large to fit all at once, and this approach isn't feasible. Instead, we use a technique called reservoir sampling.

[Reservoir sampling](#) is an algorithm for efficiently drawing random samples from a dataset  $S = \{S_1, \dots, S_N\}$  where the elements in the dataset can only be observed one at a time or in batches. In fact, reservoir sampling works even when  $N$  is not known *a priori*. If only one sample is requested, such as when  $K = 1$ , the algorithm is like this:

### Algorithm: Reservoir Sampling

- Input: dataset or data stream  $S = \{S_1, \dots, S_N\}$
- Initialize the random sample  $X = S_1$
- For each observed sample  $S_n, n = 2, \dots, N$ :
  - Pick a uniform random number  $\xi \in [0, 1]$
  - If  $\xi < 1/n$ 
    - Set  $X = S_n$
- Return  $X$

This algorithm selects a random sample such that  $P(X = S_n) = 1/N$  for all  $n = 1, \dots, N$ . When  $K > 1$  the algorithm is more complicated. Additionally, a distinction must be made between random sampling that is with and without replacement. RCF performs an augmented reservoir sampling without replacement on the training data based on the algorithms described in [2].

### Train a RCF Model and Produce Inferences

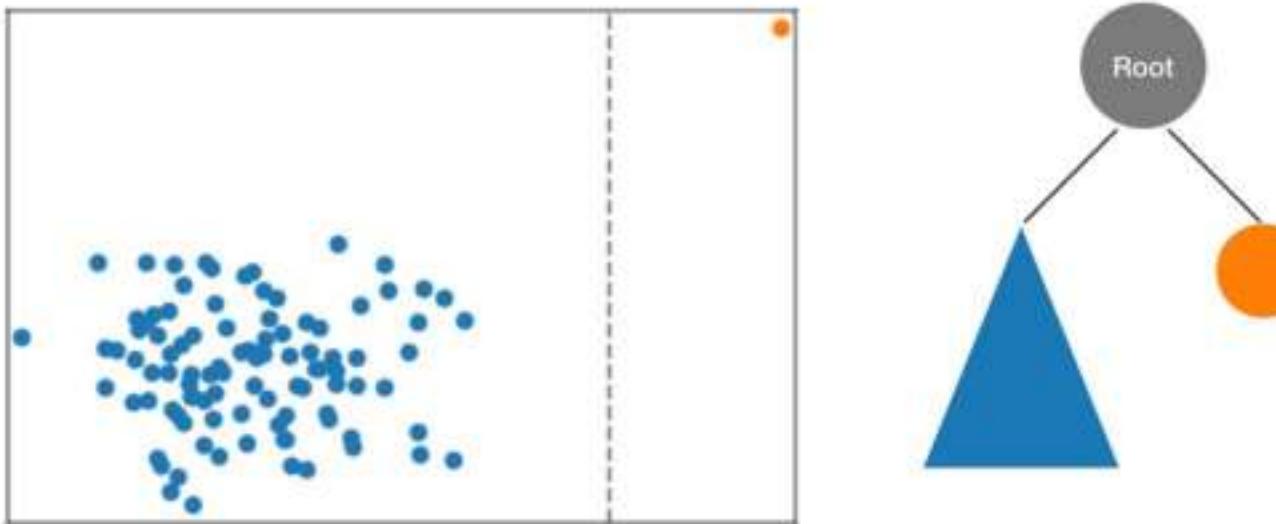
The next step in RCF is to construct a random cut forest using the random sample of data. First, the sample is partitioned into a number of equal-sized partitions equal to the number of trees in the forest. Then, each partition is sent to an individual tree. The tree recursively organizes its partition into a binary tree by partitioning the data domain into bounding boxes.

This procedure is best illustrated with an example. Suppose a tree is given the following two-dimensional dataset. The corresponding tree is initialized to the root node:



A two-dimensional dataset where the majority of data lies in a cluster (blue) except for one anomalous data point (orange). The tree is initialized with a root node.

The RCF algorithm organizes these data in a tree by first computing a bounding box of the data, selecting a random dimension (giving more weight to dimensions with higher "variance"), and then randomly determining the position of a hyperplane "cut" through that dimension. The two resulting subspaces define their own sub tree. In this example, the cut happens to separate a lone point from the remainder of the sample. The first level of the resulting binary tree consists of two nodes, one which will consist of the subtree of points to the left of the initial cut and the other representing the single point on the right.



A random cut partitioning the two-dimensional dataset. An anomalous data point is more likely to lie isolated in a bounding box at a smaller tree depth than other points.

Bounding boxes are then computed for the left and right halves of the data and the process is repeated until every leaf of the tree represents a single data point from the sample. Note that if the lone point is sufficiently far away then it is more likely that a random cut would result in point isolation. This observation provides the intuition that tree depth is, loosely speaking, inversely proportional to the anomaly score.

When performing inference using a trained RCF model the final anomaly score is reported as the average across scores reported by each tree. Note that it is often the case that the new data point does not already reside in the tree. To determine the score associated with the new point the data point is inserted into the given tree and the tree is efficiently (and temporarily) reassembled in a manner equivalent to the training process described above. That is, the resulting tree is as if the input data point were a member of the sample used to construct the tree in the first place. The reported score is inversely proportional to the depth of the input point within the tree.

### Choose Hyperparameters

The primary hyperparameters used to tune the RCF model are `num_trees` and `num_samples_per_tree`. Increasing `num_trees` has the effect of reducing the noise observed in anomaly scores since the final score is the average of the scores reported by each tree. While the optimal value is application-dependent we recommend using 100 trees to begin with as a balance between score noise and model complexity. Note that inference time is proportional to the number of trees. Although training time is also affected it is dominated by the reservoir sampling algorithm describe above.

The parameter `num_samples_per_tree` is related to the expected density of anomalies in the dataset. In particular, `num_samples_per_tree` should be chosen such that  $1/\text{num\_samples\_per\_tree}$  approximates the ratio of anomalous data to normal data. For example, if 256 samples are used in each tree then we expect our data to contain anomalies  $1/256$  or approximately 0.4% of the time. Again, an optimal value for this hyperparameter is dependent on the application.

### References

1. Sudipto Guha, Nina Mishra, Gourav Roy, and Okke Schrijvers. "Robust random cut forest based anomaly detection on streams." In *International Conference on Machine Learning*, pp. 2712-2721. 2016.

2. Byung-Hoon Park, George Ostrouchov, Nagiza F. Samatova, and Al Geist. "Reservoir-based random sampling with replacement from data stream." In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pp. 492-496. Society for Industrial and Applied Mathematics, 2004.

## RCF Hyperparameters

In the [CreateTrainingJob](#) request, you specify the training algorithm. You can also specify algorithm-specific hyperparameters as string-to-string maps. The following table lists the hyperparameters for the Amazon SageMaker RCF algorithm. For more information, including recommendations on how to choose hyperparameters, see [How RCF Works \(p. 1496\)](#).

Parameter Name	Description
<code>feature_dim</code>	<p>The number of features in the data set. (If you use the <a href="#">Random Cut Forest</a> estimator, this value is calculated for you and need not be specified.)</p> <p><b>Required</b></p> <p>Valid values: Positive integer (min: 1, max: 10000)</p>
<code>eval_metrics</code>	<p>A list of metrics used to score a labeled test data set. The following metrics can be selected for output:</p> <ul style="list-style-type: none"> <li><code>accuracy</code> - returns fraction of correct predictions.</li> <li><code>precision_recall_fscore</code> - returns the positive and negative precision, recall, and F1-scores.</li> </ul> <p><b>Optional</b></p> <p>Valid values: a list with possible values taken from <code>accuracy</code> or <code>precision_recall_fscore</code>.</p> <p>Default value: Both <code>accuracy</code>, <code>precision_recall_fscore</code> are calculated.</p>
<code>num_samples_per_tree</code>	<p>Number of random samples given to each tree from the training data set.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer (min: 1, max: 2048)</p> <p>Default value: 256</p>
<code>num_trees</code>	<p>Number of trees in the forest.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer (min: 50, max: 1000)</p> <p>Default value: 100</p>

## Tune an RCF Model

*Automatic model tuning*, also known as hyperparameter tuning or hyperparameter optimization, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset.

You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

The Amazon SageMaker RCF algorithm is an unsupervised anomaly-detection algorithm that requires a labeled test dataset for hyperparameter optimization. RCF calculates anomaly scores for test datapoints and then labels the datapoints as anomalous if their scores are beyond three standard deviations from the mean score. This is known as the three-sigma limit heuristic. The F1-score is based on the difference between calculated labels and actual labels. The hyperparameter tuning job finds the model that maximizes that score. The success of hyperparameter optimization depends on the applicability of the three-sigma limit heuristic to the test dataset.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics Computed by the RCF Algorithm

The RCF algorithm computes the following metric during training. When tuning the model, choose this metric as the objective metric.

Metric Name	Description	Optimization Direction
test:f1	F1-score on the test dataset, based on the difference between calculated labels and actual labels.	Maximize

### Tunable RCF Hyperparameters

You can tune a RCF model with the following hyperparameters.

Parameter Name	Parameter Type	Recommended Ranges
num_samples_per_tree	IntegerParameterRanges	MinValue: 1, MaxValue:2048
num_trees	IntegerParameterRanges	MinValue: 50, MaxValue:1000

### RCF Response Formats

All Amazon SageMaker built-in algorithms adhere to the common input inference format described in [Common Data Formats - Inference](#). Note that SageMaker Random Cut Forest supports both dense and sparse JSON and RecordIO formats. This topic contains a list of the available output formats for the SageMaker RCF algorithm.

#### JSON Response Format

ACCEPT: application/json.

```
{
```

```
"scores": [  
  
    {"score": 0.02},  
  
    {"score": 0.25}  
  
]  
  
}
```

### JSONLINES Response Format

ACCEPT: application/jsonlines.

```
{"score": 0.02},  
{"score": 0.25}
```

### RECORDIO Response Format

ACCEPT: application/x-recordio-protobuf.

```
[  
  
    Record = {  
  
        features = {},  
  
        label = {  
  
            'score': {  
  
                keys: [],  
  
                values: [0.25] # float32  
  
            }  
  
        }  
  
    }  
  
]
```

```
        },  
  
        Record = {  
  
            features = {},  
  
            label = {  
  
                'score': {  
  
                    keys: [],  
  
                    values: [0.23] # float32  
  
                }  
  
            }  
  
        }  
  
    ]
```

## Semantic Segmentation Algorithm

The SageMaker semantic segmentation algorithm provides a fine-grained, pixel-level approach to developing computer vision applications. It tags every pixel in an image with a class label from a predefined set of classes. Tagging is fundamental for understanding scenes, which is critical to an increasing number of computer vision applications, such as self-driving vehicles, medical imaging diagnostics, and robot sensing.

For comparison, the SageMaker [Image Classification Algorithm \(p. 1399\)](#) is a supervised learning algorithm that analyzes only whole images, classifying them into one of multiple output categories. The [Object Detection Algorithm \(p. 1479\)](#) is a supervised learning algorithm that detects and classifies all instances of an object in an image. It indicates the location and scale of each object in the image with a rectangular bounding box.

Because the semantic segmentation algorithm classifies every pixel in an image, it also provides information about the shapes of the objects contained in the image. The segmentation output is represented as a grayscale image, called a *segmentation mask*. A segmentation mask is a grayscale image with the same shape as the input image.

The SageMaker semantic segmentation algorithm is built using the [MXNet Gluon framework](#) and the [Gluon CV toolkit](#), and provides you with a choice of three build-in algorithms to train a deep neural network. You can use the [Fully-Convolutional Network \(FCN\) algorithm](#), [Pyramid Scene Parsing \(PSP\) algorithm](#), or [DeepLabV3](#).

Each of the three algorithms has two distinct components:

- The *backbone* (or *encoder*)—A network that produces reliable activation maps of features.
- The *decoder*—A network that constructs the segmentation mask from the encoded activation maps.

You also have a choice of backbones for the FCN, PSP, and DeepLabV3 algorithms: [ResNet50](#) or [ResNet101](#). These backbones include pretrained artifacts that were originally trained on the [ImageNet](#) classification task. You can fine-tune these backbones for segmentation using your own data. Or, you can initialize and train these networks from scratch using only your own data. The decoders are never pretrained.

To deploy the trained model for inference, use the SageMaker hosting service. During inference, you can request the segmentation mask either as a PNG image or as a set of probabilities for each class for each pixel. You can use these masks as part of a larger pipeline that includes additional downstream image processing or other applications.

#### Topics

- [Semantic Segmentation Sample Notebooks \(p. 1503\)](#)
- [Input/Output Interface for the Semantic Segmentation Algorithm \(p. 1503\)](#)
- [EC2 Instance Recommendation for the Semantic Segmentation Algorithm \(p. 1506\)](#)
- [Semantic Segmentation Hyperparameters \(p. 1506\)](#)
- [Tuning a Semantic Segmentation Model \(p. 1511\)](#)

## Semantic Segmentation Sample Notebooks

For a sample Jupyter notebook that uses the SageMaker semantic segmentation algorithm to train a model and deploy it to perform inferences, see the [Semantic Segmentation Example](#). For instructions on how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#).

To see a list of all of the SageMaker samples, create and open a notebook instance, and choose the [SageMaker Examples](#) tab. The example semantic segmentation notebooks are located under [Introduction to Amazon algorithms](#). To open a notebook, choose its **Use** tab, and choose **Create copy**.

## [Input/Output Interface for the Semantic Segmentation Algorithm](#)

SageMaker semantic segmentation expects the customer's training dataset to be on [Amazon Simple Storage Service \(Amazon S3\)](#). Once trained, it produces the resulting model artifacts on Amazon S3. The input interface format for the SageMaker semantic segmentation is similar to that of most standardized semantic segmentation benchmarking datasets. The dataset in Amazon S3 is expected to be presented in two channels, one for `train` and one for `validation` using four directories, two for images and two for annotations. Annotations are expected to be uncompressed PNG images. The dataset might also have a label map that describes how the annotation mappings are established. If not, the algorithm uses a default. It also supports the augmented manifest image format (`application/x-image`) for training in Pipe input mode straight from Amazon S3. For inference, an endpoint accepts images with an `image/jpg` content type.

### How Training Works

The training data is split into four directories: `train`, `train_annotation`, `validation`, and `validation_annotation`. There is a channel for each of these directories. The dataset also expected

to have one `label_map.json` file per channel for `train_annotation` and `validation_annotation` respectively. If you don't provide these JSON files, SageMaker provides the default set label map.

The dataset specifying these files should look similar to the following example:

```
s3://bucket_name
|
|- train
  |
  |- 0000.jpg
  |- coffee.jpg
|- validation
  |
  |- 00a0.jpg
  |- bananna.jpg
|- train_annotation
  |
  |- 0000.png
  |- coffee.png
|- validation_annotation
  |
  |- 00a0.png
  |- bananna.png
|- label_map
  |
  |- train_label_map.json
  |- validation_label_map.json
```

Every JPG image in the `train` and `validation` directories has a corresponding PNG label image with the same name in the `train_annotation` and `validation_annotation` directories. This naming convention helps the algorithm to associate a label with its corresponding image during training. The `train`, `train_annotation`, `validation`, and `validation_annotation` channels are mandatory. The annotations are single-channel PNG images. The format works as long as the metadata (modes) in the image helps the algorithm read the annotation images into a single-channel 8-bit unsigned integer. For more information on our support for modes, see the [Python Image Library documentation](#). We recommend using the 8-bit pixel, true color `P` mode.

The image that is encoded is a simple 8-bit integer when using modes. To get from this mapping to a map of a label, the algorithm uses one mapping file per channel, called the *label map*. The label map is used to map the values in the image with actual label indices. In the default label map, which is provided by default if you don't provide one, the pixel value in an annotation matrix (image) directly index the label. These images can be grayscale PNG files or 8-bit indexed PNG files. The label map file for the unscaled default case is the following:

```
{
  "scale": "1"
}
```

To provide some contrast for viewing, some annotation software scales the label images by a constant amount. To support this, the SageMaker semantic segmentation algorithm provides a rescaling option to scale down the values to actual label values. When scaling down doesn't convert the value to an appropriate integer, the algorithm defaults to the greatest integer less than or equal to the scale value. The following code shows how to set the scale value to rescale the label values:

```
{
  "scale": "3"
}
```

The following example shows how this `"scale"` value is used to rescale the `encoded_label` values of the input annotation image when they are mapped to the `mapped_label` values to be used in training.

The label values in the input annotation image are 0, 3, 6, with scale 3, so they are mapped to 0, 1, 2 for training:

```
encoded_label = [0, 3, 6]
mapped_label = [0, 1, 2]
```

In some cases, you might need to specify a particular color mapping for each class. Use the map option in the label mapping as shown in the following example of a `label_map` file:

```
{
  "map": {
    "0": 5,
    "1": 0,
    "2": 2
  }
}
```

This label mapping for this example is:

```
encoded_label = [0, 5, 2]
mapped_label = [1, 0, 2]
```

With label mappings, you can use different annotation systems and annotation software to obtain data without a lot of preprocessing. You can provide one label map per channel. The files for a label map in the `label_map` channel must follow the naming conventions for the four directory structure. If you don't provide a label map, the algorithm assumes a scale of 1 (the default).

### Training with the Augmented Manifest Format

The augmented manifest format enables you to do training in Pipe mode using image files without needing to create RecordIO files. The augmented manifest file contains data objects and should be in [JSON Lines](#) format, as described in the [CreateTrainingJob](#) request. Each line in the manifest is an entry containing the Amazon S3 URI for the image and the URI for the annotation image.

Each JSON object in the manifest file must contain a `source-ref` key. The `source-ref` key should contain the value of the Amazon S3 URI to the image. The labels are provided under the `AttributeNames` parameter value as specified in the [CreateTrainingJob](#) request. It can also contain additional metadata under the `metadata` tag, but these are ignored by the algorithm. In the example below, the `AttributeNames` are contained in the list of image and annotation references `[ "source-ref", "city-streets-ref" ]`. These names must have `-ref` appended to them. When using the Semantic Segmentation algorithm with Augmented Manifest, the value of the `RecordWrapperType` parameter must be `"RecordIO"` and value of the `ContentType` parameter must be `application/x-recordio`.

```
{"source-ref": "S3 bucket location", "city-streets-ref": "S3 bucket location", "city-streets-metadata": {"job-name": "label-city-streets", }}
```

For more information on augmented manifest files, see [Provide Dataset Metadata to Training Jobs with an Augmented Manifest File \(p. 1864\)](#).

### Incremental Training

You can also seed the training of a new model with a model that you trained previously using SageMaker. This incremental training saves training time when you want to train a new model with the same or similar data. Currently, incremental training is supported only for models trained with the built-in SageMaker Semantic Segmentation.

To use your own pre-trained model, specify the `ChannelName` as "model" in the `InputDataConfig` for the [CreateTrainingJob](#) request. Set the `ContentType` for the model channel to `application/x-sagemaker-model`. The `backbone`, `algorithm`, `crop_size`, and `num_classes` input parameters that define the network architecture must be consistently specified in the input hyperparameters of the new model and the pre-trained model that you upload to the model channel. For the pretrained model file, you can use the compressed (.tar.gz) artifacts from SageMaker outputs. You can only use Image formats for input data. For more information on incremental training and for instructions on how to use it, see [Incremental Training in Amazon SageMaker \(p. 1855\)](#).

### Produce Inferences

To query a trained model that is deployed to an endpoint, you need to provide an image and an `AcceptType` that denotes the type of output required. The endpoint takes JPEG images with an `image/jpeg` content type. If you request an `AcceptType` of `image/png`, the algorithm outputs a PNG file with a segmentation mask in the same format as the labels themselves. If you request an accept type of `application/x-recordio-protobuf`, the algorithm returns class probabilities encoded in recordio-protobuf format. The latter format outputs a 3D tensor where the third dimension is the same size as the number of classes. This component denotes the probability of each class label for each pixel.

## EC2 Instance Recommendation for the Semantic Segmentation Algorithm

The SageMaker semantic segmentation algorithm only supports GPU instances for training, and we recommend using GPU instances with more memory for training with large batch sizes. The algorithm can be trained using [P2/P3 EC2 Amazon Elastic Compute Cloud \(Amazon EC2\)](#) instances in single machine configurations. It supports the following GPU instances for training:

- `ml.p2.xlarge`
- `ml.p2.8xlarge`
- `ml.p2.16xlarge`
- `ml.p3.2xlarge`
- `ml.p3.8xlarge`
- `ml.p3.16xlarge`

For inference, you can use either CPU instances (such as `c5` and `m5`) and GPU instances (such as `p2` and `p3`) or both. For information about the instance types that provide varying combinations of CPU, GPU, memory, and networking capacity for inference, see [Amazon SageMaker ML Instance Types](#).

### Semantic Segmentation Hyperparameters

The following tables list the hyperparameters supported by the Amazon SageMaker semantic segmentation algorithm for network architecture, data inputs, and training. You specify Semantic Segmentation for training in the `AlgorithmName` of the [CreateTrainingJob](#) request.

#### Network Architecture Hyperparameters

Parameter Name	Description
<code>backbone</code>	<p>The backbone to use for the algorithm's encoder component.</p> <p><b>Optional</b></p> <p>Valid values: <code>resnet-50</code>, <code>resnet-101</code></p> <p>Default value: <code>resnet-50</code></p>

Parameter Name	Description
use_pretrained_model	<p>Whether a pretrained model is to be used for the backbone.</p> <p><b>Optional</b></p> <p>Valid values: <code>True</code>, <code>False</code></p> <p>Default value: <code>True</code></p>
algorithm	<p>The algorithm to use for semantic segmentation.</p> <p><b>Optional</b></p> <p>Valid values:</p> <ul style="list-style-type: none"> <li>• <code>fcn</code>: <a href="#">Fully-Convolutional Network (FCN) algorithm</a></li> <li>• <code>psp</code>: <a href="#">Pyramid Scene Parsing (PSP) algorithm</a></li> <li>• <code>deeplab</code>: <a href="#">DeepLab V3 algorithm</a></li> </ul> <p>Default value: <code>fcn</code></p>

## Data Hyperparameters

Parameter Name	Description
num_classes	<p>The number of classes to segment.</p> <p><b>Required</b></p> <p>Valid values: <math>2 \leq</math> positive integer <math>\leq 254</math></p>
num_training_samples	<p>The number of samples in the training data. The algorithm uses this value to set up the learning rate scheduler.</p> <p><b>Required</b></p> <p>Valid values: positive integer</p>
base_size	<p>Defines how images are rescaled before cropping. Images are rescaled such that the long size length is set to <code>base_size</code> multiplied by a random number from 0.5 to 2.0, and the short size is computed to preserve the aspect ratio.</p> <p><b>Optional</b></p> <p>Valid values: positive integer <math>&gt; 16</math></p> <p>Default value: 520</p>
crop_size	<p>The image size for input during training. We randomly rescale the input image based on <code>base_size</code>, and then take a random square crop with side length equal to <code>crop_size</code>. The <code>crop_size</code> will be automatically rounded up to multiples of 8.</p> <p><b>Optional</b></p> <p>Valid values: positive integer <math>&gt; 16</math></p>

Parameter Name	Description
	Default value: 240

### Training Hyperparameters

Parameter Name	Description
early_stopping	<p>Whether to use early stopping logic during training.</p> <p><b>Optional</b></p> <p>Valid values: True, False</p> <p>Default value: False</p>
early_stopping_min_epochs	<p>The minimum number of epochs that must be run.</p> <p><b>Optional</b></p> <p>Valid values: integer</p> <p>Default value: 5</p>
early_stopping_patience	<p>The number of epochs that meet the tolerance for lower performance before the algorithm enforces an early stop.</p> <p><b>Optional</b></p> <p>Valid values: integer</p> <p>Default value: 4</p>
early_stopping_tolerance	<p>If the relative improvement of the score of the training job, the mIOU, is smaller than this value, early stopping considers the epoch as not improved. This is used only when <code>early_stopping = True</code>.</p> <p><b>Optional</b></p> <p>Valid values: <math>0 \leq \text{float} \leq 1</math></p> <p>Default value: 0.0</p>
epochs	<p>The number of epochs with which to train.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 10</p>
gamma1	<p>The decay factor for the moving average of the squared gradient for <code>rmsprop</code>. Used only for <code>rmsprop</code>.</p> <p><b>Optional</b></p> <p>Valid values: <math>0 \leq \text{float} \leq 1</math></p> <p>Default value: 0.9</p>

Parameter Name	Description
gamma2	<p>The momentum factor for <code>rmsprop</code>.</p> <p><b>Optional</b></p> <p>Valid values: <math>0 \leq \text{float} \leq 1</math></p> <p>Default value: 0.9</p>
learning_rate	<p>The initial learning rate.</p> <p><b>Optional</b></p> <p>Valid values: <math>0 &lt; \text{float} \leq 1</math></p> <p>Default value: 0.001</p>
lr_scheduler	<p>The shape of the learning rate schedule that controls its decrease over time.</p> <p><b>Optional</b></p> <p>Valid values:</p> <ul style="list-style-type: none"> <li>• <code>step</code>: A stepwise decay, where the learning rate is reduced (multiplied) by the <code>lr_scheduler_factor</code> after epochs specified by <code>lr_scheduler_step</code>.</li> <li>• <code>poly</code>: A smooth decay using a polynomial function.</li> <li>• <code>cosine</code>: A smooth decay using a cosine function.</li> </ul> <p>Default value: <code>poly</code></p>
lr_scheduler_factor	<p>If <code>lr_scheduler</code> is set to <code>step</code>, the ratio by which to reduce (multipy) the <code>learning_rate</code> after each of the epochs specified by the <code>lr_scheduler_step</code>. Otherwise, ignored.</p> <p><b>Optional</b></p> <p>Valid values: <math>0 \leq \text{float} \leq 1</math></p> <p>Default value: 0.1</p>
lr_scheduler_step	<p>A comma delimited list of the epochs after which the <code>learning_rate</code> is reduced (multiplied) by an <code>lr_scheduler_factor</code>. For example, if the value is set to "10, 20", then the <code>learning_rate</code> is reduced by <code>lr_scheduler_factor</code> after the 10th epoch and again by this factor after 20th epoch.</p> <p><b>Conditionally Required</b> if <code>lr_scheduler</code> is set to <code>step</code>. Otherwise, ignored.</p> <p>Valid values: string</p> <p>Default value: (No default, as the value is required when used.)</p>

Parameter Name	Description
mini_batch_size	<p>The batch size for training. Using a large <code>mini_batch_size</code> usually results in faster training, but it might cause you to run out of memory. Memory usage is affected by the values of the <code>mini_batch_size</code> and <code>image_shape</code> parameters, and the backbone architecture.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 16</p>
momentum	<p>The momentum for the <code>sgd</code> optimizer. When you use other optimizers, the semantic segmentation algorithm ignores this parameter.</p> <p><b>Optional</b></p> <p>Valid values: <math>0 &lt; \text{float} \leq 1</math></p> <p>Default value: 0.9</p>
optimizer	<p>The type of optimizer. For more information about an optimizer, choose the appropriate link:</p> <ul style="list-style-type: none"> <li>• <code>adam</code>: <a href="#">Adaptive momentum estimation</a></li> <li>• <code>adagrad</code>: <a href="#">Adaptive gradient descent</a></li> <li>• <code>nag</code>: <a href="#">Nesterov accelerated gradient</a></li> <li>• <code>rmsprop</code>: <a href="#">Root mean square propagation</a></li> <li>• <code>sgd</code>: <a href="#">Stochastic gradient descent</a></li> </ul> <p><b>Optional</b></p> <p>Valid values: <code>adam</code>, <code>adagrad</code>, <code>nag</code>, <code>rmsprop</code>, <code>sgd</code></p> <p>Default value: <code>sgd</code></p>
syncbn	<p>If set to <code>True</code>, the batch normalization mean and variance are computed over all the samples processed across the GPUs.</p> <p><b>Optional</b></p> <p>Valid values: <code>True</code>, <code>False</code></p> <p>Default value: <code>False</code></p>

Parameter Name	Description
validation_mini_batch	<p>The batch size for validation. A large <code>mini_batch_size</code> usually results in faster training, but it might cause you to run out of memory. Memory usage is affected by the values of the <code>mini_batch_size</code> and <code>image_shape</code> parameters, and the backbone architecture.</p> <ul style="list-style-type: none"> <li>To score the validation on the entire image without cropping the images, set this parameter to 1. Use this option if you want to measure performance on the entire image as a whole.</li> </ul> <p><b>Note</b> Setting the <code>validation_mini_batch_size</code> parameter to 1 causes the algorithm to create a new network model for every image. This might slow validation and training.</p> <ul style="list-style-type: none"> <li>To crop images to the size specified in the <code>crop_size</code> parameter, even during evaluation, set this parameter to a value greater than 1.</li> </ul> <p><b>Optional</b> Valid values: positive integer Default value: 16</p>
weight_decay	<p>The weight decay coefficient for the <code>sgd</code> optimizer. When you use other optimizers, the algorithm ignores this parameter.</p> <p><b>Optional</b> Valid values: <math>0 &lt; \text{float} &lt; 1</math> Default value: 0.0001</p>

## Tuning a Semantic Segmentation Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

### Metrics Computed by the Semantic Segmentation Algorithm

The semantic segmentation algorithm reports two validation metrics. When tuning hyperparameter values, choose one of these metrics as the objective.

Metric Name	Description	Optimization Direction
validation:mIOU	The area of the intersection of the predicted segmentation and the ground truth divided by the area of union between them for images in the validation set. Also known as the Jaccard Index.	Maximize
validation:pixel_accuracy	The percentage of pixels that are correctly classified in images from the validation set.	Maximize

## Tunable Semantic Segmentation Hyperparameters

You can tune the following hyperparameters for the semantic segmentation algorithm.

Parameter Name	Parameter Type	Recommended Ranges
learning_rate	ContinuousParameterRange	MinValue: 1e-4, MaxValue: 1e-1
mini_batch_size	IntegerParameterRanges	MinValue: 1, MaxValue: 128
momentum	ContinuousParameterRange	MinValue: 0.9, MaxValue: 0.999
optimzer	CategoricalParameterRanges	['sgd', 'adam', 'adadelta']
weight_decay	ContinuousParameterRange	MinValue: 1e-5, MaxValue: 1e-3

## Sequence-to-Sequence Algorithm

Amazon SageMaker Sequence to Sequence is a supervised learning algorithm where the input is a sequence of tokens (for example, text, audio) and the output generated is another sequence of tokens. Example applications include: machine translation (input a sentence from one language and predict what that sentence would be in another language), text summarization (input a longer string of words and predict a shorter string of words that is a summary), speech-to-text (audio clips converted into output sentences in tokens). Recently, problems in this domain have been successfully modeled with deep neural networks that show a significant performance boost over previous methodologies. Amazon SageMaker seq2seq uses Recurrent Neural Networks (RNNs) and Convolutional Neural Network (CNN) models with attention as encoder-decoder architectures.

### Topics

- [Input/Output Interface for the Sequence-to-Sequence Algorithm \(p. 1512\)](#)
- [EC2 Instance Recommendation for the Sequence-to-Sequence Algorithm \(p. 1513\)](#)
- [Sequence-to-Sequence Sample Notebooks \(p. 1513\)](#)
- [How Sequence-to-Sequence Works \(p. 1514\)](#)
- [Sequence-to-Sequence Hyperparameters \(p. 1514\)](#)
- [Tune a Sequence-to-Sequence Model \(p. 1522\)](#)

## Input/Output Interface for the Sequence-to-Sequence Algorithm

### Training

SageMaker seq2seq expects data in RecordIO-Protobuf format. However, the tokens are expected as integers, not as floating points, as is usually the case.

A script to convert data from tokenized text files to the protobuf format is included in [the seq2seq example notebook](#). In general, it packs the data into 32-bit integer tensors and generates the necessary vocabulary files, which are needed for metric calculation and inference.

After preprocessing is done, the algorithm can be invoked for training. The algorithm expects three channels:

- **train:** It should contain the training data (for example, the `train.rec` file generated by the preprocessing script).

- validation: It should contain the validation data (for example, the `val.rec` file generated by the preprocessing script).
- vocab: It should contain two vocabulary files (`vocab.src.json` and `vocab.trg.json`)

If the algorithm doesn't find data in any of these three channels, training results in an error.

### Inference

For hosted endpoints, inference supports two data formats. To perform inference using space separated text tokens, use the `application/json` format. Otherwise, use the `recordio-protobuf` format to work with the integer encoded data. Both modes support batching of input data. `application/json` format also allows you to visualize the attention matrix.

- `application/json`: Expects the input in JSON format and returns the output in JSON format. Both content and accept types should be `application/json`. Each sequence is expected to be a string with whitespace separated tokens. This format is recommended when the number of source sequences in the batch is small. It also supports the following additional configuration options:

`configuration: {attention_matrix: true}`: Returns the attention matrix for the particular input sequence.

- `application/x-recordio-protobuf`: Expects the input in `recordio-protobuf` format and returns the output in `recordio-protobuf` format. Both content and accept types should be `applications/x-recordio-protobuf`. For this format, the source sequences must be converted into a list of integers for subsequent protobuf encoding. This format is recommended for bulk inference.

For batch transform, inference supports JSON Lines format. Batch transform expects the input in JSON Lines format and returns the output in JSON Lines format. Both content and accept types should be `application/jsonlines`. The format for input is as follows:

```
content-type: application/jsonlines

{"source": "source_sequence_0"}
{"source": "source_sequence_1"}
```

The format for response is as follows:

```
accept: application/jsonlines

{"target": "predicted_sequence_0"}
{"target": "predicted_sequence_1"}
```

For additional details on how to serialize and deserialize the inputs and outputs to specific formats for inference, see the [Sequence-to-Sequence Sample Notebooks \(p. 1513\)](#).

## EC2 Instance Recommendation for the Sequence-to-Sequence Algorithm

Currently Amazon SageMaker seq2seq is only supported on GPU instance types and is only set up to train on a single machine. But it does also offer support for multiple GPUs.

### Sequence-to-Sequence Sample Notebooks

For a sample notebook that shows how to use the SageMaker Sequence to Sequence algorithm to train a English-German translation model, see [Machine Translation English-German Example Using SageMaker Seq2Seq](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the

SageMaker samples. The topic modeling example notebooks using the NTM algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

## How Sequence-to-Sequence Works

Typically, a neural network for sequence-to-sequence modeling consists of a few layers, including:

- An **embedding layer**. In this layer, the input matrix, which is input tokens encoded in a sparse way (for example, one-hot encoded) are mapped to a dense feature layer. This is required because a high-dimensional feature vector is more capable of encoding information regarding a particular token (word for text corpora) than a simple one-hot-encoded vector. It is also a standard practice to initialize this embedding layer with a pre-trained word vector like [FastText](#) or [Glove](#) or to initialize it randomly and learn the parameters during training.
- An **encoder layer**. After the input tokens are mapped into a high-dimensional feature space, the sequence is passed through an encoder layer to compress all the information from the input embedding layer (of the entire sequence) into a fixed-length feature vector. Typically, an encoder is made of RNN-type networks like long short-term memory (LSTM) or gated recurrent units (GRU). ([Colah's blog](#) explains LSTM in a great detail.)
- A **decoder layer**. The decoder layer takes this encoded feature vector and produces the output sequence of tokens. This layer is also usually built with RNN architectures (LSTM and GRU).

The whole model is trained jointly to maximize the probability of the target sequence given the source sequence. This model was first introduced by [Sutskever et al.](#) in 2014.

**Attention mechanism.** The disadvantage of an encoder-decoder framework is that model performance decreases as and when the length of the source sequence increases because of the limit of how much information the fixed-length encoded feature vector can contain. To tackle this problem, in 2015, Bahdanau et al. proposed the [attention mechanism](#). In an attention mechanism, the decoder tries to find the location in the encoder sequence where the most important information could be located and uses that information and previously decoded words to predict the next token in the sequence.

For more in details, see the whitepaper [Effective Approaches to Attention-based Neural Machine Translation](#) by Luong, et al. that explains and simplifies calculations for various attention mechanisms. Additionally, the whitepaper [Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation](#) by Wu, et al. describes Google's architecture for machine translation, which uses skip connections between encoder and decoder layers.

## Sequence-to-Sequence Hyperparameters

Parameter Name	Description
<code>batch_size</code>	<p>Mini batch size for gradient descent.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 64</p>
<code>beam_size</code>	<p>Length of the beam for beam search. Used during training for computing <code>bleu</code> and used during inference.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 5</p>

Parameter Name	Description
bleu_sample_size	<p>Number of instances to pick from validation dataset to decode and compute bleu score during training. Set to -1 to use full validation set (if bleu is chosen as optimized_metric).</p> <p><b>Optional</b></p> <p>Valid values: integer</p> <p>Default value: 0</p>
bucket_width	<p>Returns (source,target) buckets up to (max_seq_len_source,max_seq_len_target). The longer side of the data uses steps of bucket_width while the shorter side uses steps scaled down by the average target/source length ratio. If one sided reaches its maximum length before the other, width of extra buckets on that side is fixed to that side of max_len.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 10</p>
bucketing_enabled	<p>Set to false to disable bucketing, unroll to maximum length.</p> <p><b>Optional</b></p> <p>Valid values: true or false</p> <p>Default value: true</p>
checkpoint_frequency_num_batches	<p>Checkpoint and evaluate every x batches. This checkpointing hyperparameter is passed to the SageMaker's seq2seq algorithm for early stopping and retrieving the best model. The algorithm's checkpointing runs locally in the algorithm's training container and is not compatible with SageMaker checkpointing. The algorithm temporarily saves checkpoints to a local path and stores the best model artifact to the model output path in S3 after the training job has stopped.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 1000</p>

Parameter Name	Description
<code>checkpoint_threshold</code>	<p>Maximum number of checkpoints model is allowed to not improve in <code>optimized_metric</code> on validation dataset before training is stopped. This checkpointing hyperparameter is passed to the SageMaker's seq2seq algorithm for early stopping and retrieving the best model. The algorithm's checkpointing runs locally in the algorithm's training container and is not compatible with SageMaker checkpointing. The algorithm temporarily saves checkpoints to a local path and stores the best model artifact to the model output path in S3 after the training job has stopped.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 3</p>
<code>clip_gradient</code>	<p>Clip absolute gradient values greater than this. Set to negative to disable.</p> <p><b>Optional</b></p> <p>Valid values: float</p> <p>Default value: 1</p>
<code>cnn_activation_type</code>	<p>The <code>cnn</code> activation type to be used.</p> <p><b>Optional</b></p> <p>Valid values: String. One of <code>glu</code>, <code>relu</code>, <code>softrelu</code>, <code>sigmoid</code>, or <code>tanh</code>.</p> <p>Default value: <code>glu</code></p>
<code>cnn_hidden_dropout</code>	<p>Dropout probability for dropout between convolutional layers.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range in [0,1].</p> <p>Default value: 0</p>
<code>cnn_kernel_width_decoder</code>	<p>Kernel width for the <code>cnn</code> decoder.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 5</p>

Parameter Name	Description
cnn_kernel_width_encoder	<p>Kernel width for the <code>cnn</code> encoder.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 3</p>
cnn_num_hidden	<p>Number of <code>cnn</code> hidden units for encoder and decoder.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 512</p>
decoder_type	<p>Decoder type.</p> <p><b>Optional</b></p> <p>Valid values: String. Either <code>rnn</code> or <code>cnn</code>.</p> <p>Default value: <code>rnn</code></p>
embed_dropout_source	<p>Dropout probability for source side embeddings.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range in [0,1].</p> <p>Default value: 0</p>
embed_dropout_target	<p>Dropout probability for target side embeddings.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range in [0,1].</p> <p>Default value: 0</p>
encoder_type	<p>Encoder type. The <code>rnn</code> architecture is based on attention mechanism by Bahdanau et al. and <code>cnn</code> architecture is based on Gehring et al.</p> <p><b>Optional</b></p> <p>Valid values: String. Either <code>rnn</code> or <code>cnn</code>.</p> <p>Default value: <code>rnn</code></p>
fixed_rate_lr_half_life	<p>Half life for learning rate in terms of number of checkpoints for <code>fixed_rate_*</code> schedulers.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 10</p>

Parameter Name	Description
<code>learning_rate</code>	<p>Initial learning rate.</p> <p><b>Optional</b></p> <p>Valid values: float</p> <p>Default value: 0.0003</p>
<code>loss_type</code>	<p>Loss function for training.</p> <p><b>Optional</b></p> <p>Valid values: String. <code>cross-entropy</code></p> <p>Default value: <code>cross-entropy</code></p>
<code>lr_scheduler_type</code>	<p>Learning rate scheduler type. <code>plateau_reduce</code> means reduce the learning rate whenever <code>optimized_metric</code> on <code>validation_accuracy</code> plateaus. <code>inv_t</code> is inverse time decay. <math>\text{learning\_rate}/(1+\text{decay\_rate}^t)</math></p> <p><b>Optional</b></p> <p>Valid values: String. One of <code>plateau_reduce</code>, <code>fixed_rate_inv_t</code>, or <code>fixed_rate_inv_sqrt_t</code>.</p> <p>Default value: <code>plateau_reduce</code></p>
<code>max_num_batches</code>	<p>Maximum number of updates/batches to process. -1 for infinite.</p> <p><b>Optional</b></p> <p>Valid values: integer</p> <p>Default value: -1</p>
<code>max_num_epochs</code>	<p>Maximum number of epochs to pass through training data before fitting is stopped. Training continues until this number of epochs even if validation accuracy is not improving if this parameter is passed. Ignored if not passed.</p> <p><b>Optional</b></p> <p>Valid values: Positive integer and less than or equal to <code>max_num_epochs</code>.</p> <p>Default value: none</p>
<code>max_seq_len_source</code>	<p>Maximum length for the source sequence length. Sequences longer than this length are truncated to this length.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 100</p>

Parameter Name	Description
max_seq_len_target	<p>Maximum length for the target sequence length. Sequences longer than this length are truncated to this length.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 100</p>
min_num_epochs	<p>Minimum number of epochs the training must run before it is stopped via <code>early_stopping</code> conditions.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 0</p>
momentum	<p>Momentum constant used for <code>sgd</code>. Don't pass this parameter if you are using <code>adam</code> or <code>rmsprop</code>.</p> <p><b>Optional</b></p> <p>Valid values: float</p> <p>Default value: none</p>
num_embed_source	<p>Embedding size for source tokens.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 512</p>
num_embed_target	<p>Embedding size for target tokens.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 512</p>
num_layers_decoder	<p>Number of layers for Decoder <code>rnn</code> or <code>cnn</code>.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 1</p>
num_layers_encoder	<p>Number of layers for Encoder <code>rnn</code> or <code>cnn</code>.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 1</p>

Parameter Name	Description
optimized_metric	<p>Metrics to optimize with early stopping.</p> <p><b>Optional</b></p> <p>Valid values: String. One of <code>perplexity</code>, <code>accuracy</code>, or <code>bleu</code>.</p> <p>Default value: <code>perplexity</code></p>
optimizer_type	<p>Optimizer to choose from.</p> <p><b>Optional</b></p> <p>Valid values: String. One of <code>adam</code>, <code>sgd</code>, or <code>rmsprop</code>.</p> <p>Default value: <code>adam</code></p>
plateau_reduce_lr_factor	<p>Factor to multiply learning rate with (for <code>plateau_reduce</code>).</p> <p><b>Optional</b></p> <p>Valid values: float</p> <p>Default value: 0.5</p>
plateau_reduce_lr_threshold	<p>For <code>plateau_reduce</code> scheduler, multiply learning rate with reduce factor if <code>optimized_metric</code> didn't improve for this many checkpoints.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 3</p>
rnn_attention_in_upper_layers	<p>Pass the attention to upper layers of <code>rnn</code>, like Google NMT paper. Only applicable if more than one layer is used.</p> <p><b>Optional</b></p> <p>Valid values: boolean (<code>true</code> or <code>false</code>)</p> <p>Default value: <code>true</code></p>
rnn_attention_num_hidden	<p>Number of hidden units for attention layers. defaults to <code>rnn_num_hidden</code>.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: <code>rnn_num_hidden</code></p>

Parameter Name	Description
rnn_attention_type	<p>Attention model for encoders. <code>mlp</code> refers to concat and bilinear refers to general from the Luong et al. paper.</p> <p><b>Optional</b></p> <p>Valid values: String. One of <code>dot</code>, <code>fixed</code>, <code>mlp</code>, or <code>bilinear</code>.</p> <p>Default value: <code>mlp</code></p>
rnn_cell_type	<p>Specific type of <code>rnn</code> architecture.</p> <p><b>Optional</b></p> <p>Valid values: String. Either <code>lstm</code> or <code>gru</code>.</p> <p>Default value: <code>lstm</code></p>
rnn_decoder_state_init	<p>How to initialize <code>rnn</code> decoder states from encoders.</p> <p><b>Optional</b></p> <p>Valid values: String. One of <code>last</code>, <code>avg</code>, or <code>zero</code>.</p> <p>Default value: <code>last</code></p>
rnn_first_residual_layer	<p>First <code>rnn</code> layer to have a residual connection, only applicable if number of layers in encoder or decoder is more than 1.</p> <p><b>Optional</b></p> <p>Valid values: positive integer</p> <p>Default value: 2</p>
rnn_num_hidden	<p>The number of <code>rnn</code> hidden units for encoder and decoder. This must be a multiple of 2 because the algorithm uses bi-directional Long Term Short Term Memory (LSTM) by default.</p> <p><b>Optional</b></p> <p>Valid values: positive even integer</p> <p>Default value: 1024</p>
rnn_residual_connections	<p>Add residual connection to stacked <code>rnn</code>. Number of layers should be more than 1.</p> <p><b>Optional</b></p> <p>Valid values: boolean (<code>true</code> or <code>false</code>)</p> <p>Default value: <code>false</code></p>

Parameter Name	Description
<code>rnn_decoder_hidden_dropout</code>	<p>Dropout probability for hidden state that combines the context with the <i>rnn</i> hidden state in the decoder.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range in [0,1].</p> <p>Default value: 0</p>
<code>training_metric</code>	<p>Metrics to track on training on validation data.</p> <p><b>Optional</b></p> <p>Valid values: String. Either <code>perplexity</code> or <code>accuracy</code>.</p> <p>Default value: <code>perplexity</code></p>
<code>weight_decay</code>	<p>Weight decay constant.</p> <p><b>Optional</b></p> <p>Valid values: float</p> <p>Default value: 0</p>
<code>weight_init_scale</code>	<p>Weight initialization scale (for <code>uniform</code> and <code>xavier</code> initialization).</p> <p><b>Optional</b></p> <p>Valid values: float</p> <p>Default value: 2.34</p>
<code>weight_init_type</code>	<p>Type of weight initialization.</p> <p><b>Optional</b></p> <p>Valid values: String. Either <code>uniform</code> or <code>xavier</code>.</p> <p>Default value: <code>xavier</code></p>
<code>xavier_factor_type</code>	<p>Xavier factor type.</p> <p><b>Optional</b></p> <p>Valid values: String. One of <code>in</code>, <code>out</code>, or <code>avg</code>.</p> <p>Default value: <code>in</code></p>

## Tune a Sequence-to-Sequence Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your dataset. You choose the tunable hyperparameters, a range of values for each, and an objective metric. You choose the objective metric from the metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the objective metric.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics Computed by the Sequence-to-Sequence Algorithm

The sequence to sequence algorithm reports three metrics that are computed during training. Choose one of them as an objective to optimize when tuning the hyperparameter values.

Metric Name	Description	Optimization Direction
validation:accuracy	Accuracy computed on the validation dataset.	Maximize
validation:bleu	Bleu score computed on the validation dataset. Because BLEU computation is expensive, you can choose to compute BLEU on a random subsample of the validation dataset to speed up the overall training process. Use the <code>bleu_sample_size</code> parameter to specify the subsample.	Maximize
validation:perplexity	Perplexity, is a loss function computed on the validation dataset. Perplexity measures the cross-entropy between an empirical sample and the distribution predicted by a model and so provides a measure of how well a model predicts the sample values, Models that are good at predicting a sample have a low perplexity.	Minimize

### Tunable Sequence-to-Sequence Hyperparameters

You can tune the following hyperparameters for the SageMaker Sequence to Sequence algorithm. The hyperparameters that have the greatest impact on sequence to sequence objective metrics are: `batch_size`, `optimizer_type`, `learning_rate`, `num_layers_encoder`, and `num_layers_decoder`.

Parameter Name	Parameter Type	Recommended Ranges
<code>num_layers_encoder</code>	IntegerParameterRange	[1-10]
<code>num_layers_decoder</code>	IntegerParameterRange	[1-10]
<code>batch_size</code>	CategoricalParameterRange	[16,32,64,128,256,512,1024,2048]
<code>optimizer_type</code>	CategoricalParameterRange	['adam', 'sgd', 'rmsprop']
<code>weight_init_type</code>	CategoricalParameterRange	['xavier', 'uniform']
<code>weight_init_scale</code>	ContinuousParameterRange	For the xavier type: MinValue: 2.0, MaxValue: 3.0 For the uniform type: MinValue: -1.0, MaxValue: 1.0
<code>learning_rate</code>	ContinuousParameterRange	MinValue: 0.00005, MaxValue: 0.2
<code>weight_decay</code>	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.1

Parameter Name	Parameter Type	Recommended Ranges
momentum	ContinuousParameterRange	MinValue: 0.5, MaxValue: 0.9
clip_gradient	ContinuousParameterRange	MinValue: 1.0, MaxValue: 5.0
rnn_num_hidden	CategoricalParameterRange	Applicable only to recurrent neural networks (RNNs). [128,256,512,1024,2048]
cnn_num_hidden	CategoricalParameterRange	Applicable only to convolutional neural networks (CNNs). [128,256,512,1024,2048]
num_embed_source	IntegerParameterRange	[256-512]
num_embed_target	IntegerParameterRange	[256-512]
embed_dropout_source	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.5
embed_dropout_target	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.5
rnn_decoder_hidden_dropout	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.5
cnn_hidden_dropout	ContinuousParameterRange	MinValue: 0.0, MaxValue: 0.5
lr_scheduler_type	CategoricalParameterRange	['plateau_reduce', 'fixed_rate_inv_t', 'fixed_rate_inv_sqrt_t']
plateau_reduce_lr_factor	ContinuousParameterRange	MinValue: 0.1, MaxValue: 0.5
plateau_reduce_lr_threshold	IntegerParameterRange	[1-5]
fixed_rate_lr_half_life	IntegerParameterRange	[10-30]

## XGBoost Algorithm

The [XGBoost](#) (eXtreme Gradient Boosting) is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm that attempts to accurately predict a target variable by combining an ensemble of estimates from a set of simpler and weaker models. The XGBoost algorithm performs well in machine learning competitions because of its robust handling of a variety of data types, relationships, distributions, and the variety of hyperparameters that you can fine-tune. You can use XGBoost for regression, classification (binary and multiclass), and ranking problems.

You can use the new release of the XGBoost algorithm either as a Amazon SageMaker built-in algorithm or as a framework to run training scripts in your local environments. This implementation has a smaller memory footprint, better logging, improved hyperparameter validation, and an expanded set of metrics than the original versions. It provides an XGBoost estimator that executes a training script in a

managed XGBoost environment. The current release of SageMaker XGBoost is based on the original XGBoost versions 0.90, 1.0, and 1.2.

## Supported versions

- Framework (open source) mode: 0.90-1, 0.90-2, 1.0-1, 1.2-1, 1.2-2, 1.3-1
- Algorithm mode: 0.90-1, 0.90-2, 1.0-1, 1.2-1, 1.2-2, 1.3-1

### Note

XGBoost 1.1 is not supported on SageMaker because XGBoost 1.1 has a broken capability to run prediction when the test input has fewer features than the training data in LIBSVM inputs. This capability has been restored in XGBoost 1.2. Consider using SageMaker XGBoost 1.2-2 or later.

## How to Use SageMaker XGBoost

With SageMaker, you can use XGBoost as a built-in algorithm or framework. By using XGBoost as a framework, you have more flexibility and access to more advanced scenarios, such as k-fold cross-validation, because you can customize your own training scripts.

### • Use XGBoost as a framework

Use XGBoost as a framework to run your customized training scripts that can incorporate additional data processing into your training jobs. In the following code example, you can find how SageMaker Python SDK provides the XGBoost API as a framework in the same way it provides other framework APIs, such as TensorFlow, MXNet, and PyTorch.

SageMaker Python SDK v1

```
import boto3
import sagemaker
from sagemaker.xgboost.estimator import XGBoost
from sagemaker.session import s3_input, Session

# initialize hyperparameters
hyperparameters = {
    "max_depth": "5",
    "eta": "0.2",
    "gamma": "4",
    "min_child_weight": "6",
    "subsample": "0.7",
    "verbosity": "1",
    "objective": "reg:linear",
    "num_round": "50"}

# set an output path where the trained model will be saved
bucket = sagemaker.Session().default_bucket()
prefix = 'DEMO-xgboost-as-a-framework'
output_path = 's3://{}//{}//{}//{}//output'.format(bucket, prefix, 'abalone-xgb-framework')

# construct a SageMaker XGBoost estimator
# specify the entry_point to your xgboost training script
estimator = XGBoost(entry_point = "your_xgboost_abalone_script.py",
                    framework_version='1.2-2',
                    hyperparameters=hyperparameters,
                    role=sagemaker.get_execution_role(),
                    instance_count=1,
                    instance_type='ml.m5.2xlarge',
                    output_path=output_path)

# define the data type and paths to the training and validation datasets
content_type = "libsvm"
```

```

train_input = s3_input("s3://{}:{}{}".format(bucket, prefix, 'train'),
                      content_type=content_type)
validation_input = s3_input("s3://{}:{}{}".format(bucket, prefix, 'validation'),
                           content_type=content_type)

# execute the XGBoost training job
estimator.fit({'train': train_input, 'validation': validation_input})

```

### SageMaker Python SDK v2

```

import boto3
import sagemaker
from sagemaker.xgboost.estimator import XGBoost
from sagemaker.session import Session
from sagemaker.inputs import TrainingInput

# initialize hyperparameters
hyperparameters = {
    "max_depth": "5",
    "eta": "0.2",
    "gamma": "4",
    "min_child_weight": "6",
    "subsample": "0.7",
    "verbosity": "1",
    "objective": "reg:linear",
    "num_round": "50"}

# set an output path where the trained model will be saved
bucket = sagemaker.Session().default_bucket()
prefix = 'DEMO-xgboost-as-a-framework'
output_path = 's3://{}:{}{}/output'.format(bucket, prefix, 'abalone-xgb-framework')

# construct a SageMaker XGBoost estimator
# specify the entry_point to your xgboost training script
estimator = XGBoost(entry_point = "your_xgboost_abalone_script.py",
                     framework_version='1.2-2',
                     hyperparameters=hyperparameters,
                     role=sagemaker.get_execution_role(),
                     instance_count=1,
                     instance_type='ml.m5.2xlarge',
                     output_path=output_path)

# define the data type and paths to the training and validation datasets
content_type = "libsvm"
train_input = TrainingInput("s3://{}:{}{}".format(bucket, prefix, 'train'),
                           content_type=content_type)
validation_input = TrainingInput("s3://{}:{}{}".format(bucket, prefix,
                                                     'validation'), content_type=content_type)

# execute the XGBoost training job
estimator.fit({'train': train_input, 'validation': validation_input})

```

For an end-to-end example of using SageMaker XGBoost as a framework, see [Regression with Amazon SageMaker XGBoost](#)

- **Use XGBoost as a built-in algorithm**

Use the XGBoost built-in algorithm to build an XGBoost training container as shown in the following code example. You can automatically spot the XGBoost built-in algorithm image URI using the SageMaker `image_uris.retrieve` API (or the `get_image_uri` API if using [Amazon SageMaker Python SDK](#) version 1). If you want to ensure if the `image_uris.retrieve` API finds the correct URI, see [Common parameters for built-in algorithms](#) and look up `xgboost` from the full list of built-in algorithm image URIs and available regions.

After specifying the XGBoost image URI, you can use the XGBoost container to construct an estimator using the SageMaker Estimator API and initiate a training job. This XGBoost built-in algorithm mode does not incorporate your own XGBoost training script and runs directly on the input datasets.

#### SageMaker Python SDK v1

```

import sagemaker
import boto3
from sagemaker.amazon.amazon_estimator import get_image_uri
from sagemaker.session import s3_input, Session

# initialize hyperparameters
hyperparameters = {
    "max_depth": "5",
    "eta": "0.2",
    "gamma": "4",
    "min_child_weight": "6",
    "subsample": "0.7",
    "objective": "reg:squarederror",
    "num_round": "50"}

# set an output path where the trained model will be saved
bucket = sagemaker.Session().default_bucket()
prefix = 'DEMO-xgboost-as-a-built-in-algo'
output_path = 's3://{}//{}//{}//output'.format(bucket, prefix, 'abalone-xgb-built-in-algo')

# this line automatically looks for the XGBoost image URI and builds an XGBoost
# container.
# specify the repo_version depending on your preference.
xgboost_container = get_image_uri(boto3.Session().region_name,
                                   'xgboost',
                                   repo_version='1.2-2')

# construct a SageMaker estimator that calls the xgboost-container
estimator = sagemaker.estimator.Estimator(image_name=xgboost_container,
                                           hyperparameters=hyperparameters,
                                           role=sagemaker.get_execution_role(),
                                           instance_count=1,
                                           instance_type='ml.m5.2xlarge',
                                           train_volume_size=5, # 5 GB
                                           output_path=output_path)

# define the data type and paths to the training and validation datasets
content_type = "libsvm"
train_input = s3_input("s3://{}//{}//{}//".format(bucket, prefix, 'train'),
                      content_type=content_type)
validation_input = s3_input("s3://{}//{}//{}//".format(bucket, prefix, 'validation'),
                           content_type=content_type)

# execute the XGBoost training job
estimator.fit({'train': train_input, 'validation': validation_input})

```

#### SageMaker Python SDK v2

```

import sagemaker
import boto3
from sagemaker import image_uris
from sagemaker.session import Session
from sagemaker.inputs import TrainingInput

# initialize hyperparameters
hyperparameters = {

```

```

    "max_depth": "5",
    "eta": "0.2",
    "gamma": "4",
    "min_child_weight": "6",
    "subsample": "0.7",
    "objective": "reg:squarederror",
    "num_round": "50"}
```

```

# set an output path where the trained model will be saved
bucket = sagemaker.Session().default_bucket()
prefix = 'DEMO-xgboost-as-a-built-in-algo'
output_path = 's3://{}//{}//{}//output'.format(bucket, prefix, 'abalone-xgb-built-in-algo')

# this line automatically looks for the XGBoost image URI and builds an XGBoost
# container.
# specify the repo_version depending on your preference.
xgboost_container = sagemaker.image_uris.retrieve("xgboost", region, "1.2-2")

# construct a SageMaker estimator that calls the xgboost-container
estimator = sagemaker.estimator.Estimator(image_uri=xgboost_container,
                                            hyperparameters=hyperparameters,
                                            role=sagemaker.get_execution_role(),
                                            instance_count=1,
                                            instance_type='ml.m5.2xlarge',
                                            volume_size=5, # 5 GB
                                            output_path=output_path)

# define the data type and paths to the training and validation datasets
content_type = "libsvm"
train_input = TrainingInput("s3://{}//{}//{}//".format(bucket, prefix, 'train'),
                           content_type=content_type)
validation_input = TrainingInput("s3://{}//{}//{}//".format(bucket, prefix,
                           'validation'), content_type=content_type)

# execute the XGBoost training job
estimator.fit({'train': train_input, 'validation': validation_input})
```

For more information about how to set up the XGBoost as a built-in algorithm, see the following notebook examples.

- [Managed Spot Training for XGBoost](#)
- [Regression with Amazon SageMaker XGBoost \(Parquet input\)](#)

## Input/Output Interface for the XGBoost Algorithm

Gradient boosting operates on tabular data, with the rows representing observations, one column representing the target variable or label, and the remaining columns representing features.

The SageMaker implementation of XGBoost supports CSV and libsvm formats for training and inference:

- For Training ContentType, valid inputs are *text/libsvm* (default) or *text/csv*.
- For Inference ContentType, valid inputs are *text/libsvm* (default) or *text/csv*.

### Note

For CSV training, the algorithm assumes that the target variable is in the first column and that the CSV does not have a header record.

For CSV inference, the algorithm assumes that CSV input does not have the label column.

For libsvm training, the algorithm assumes that the label is in the first column. Subsequent columns contain the zero-based index value pairs for features. So each row has the format:

<label> <index0>:<value0> <index1>:<value1> ... Inference requests for libsvm might not have labels in the libsvm format.

This differs from other SageMaker algorithms, which use the protobuf training input format to maintain greater consistency with standard XGBoost data formats.

For CSV training input mode, the total memory available to the algorithm (Instance Count \* the memory available in the `InstanceType`) must be able to hold the training dataset. For libsvm training input mode, it's not required, but we recommend it.

SageMaker XGBoost uses the Python pickle module to serialize/deserialize the model, which can be used for saving/loading the model.

### To use a model trained with SageMaker XGBoost in open source XGBoost

- Use the following Python code:

```
import pickle as pkl
import tarfile

t = tarfile.open('model.tar.gz', 'r:gz')
t.extractall()

model = pkl.load(open(model_file_path, 'rb'))

# prediction with test data
pred = model.predict(dtest)
```

### To differentiate the importance of labelled data points use Instance Weight Supports

- SageMaker XGBoost allows customers to differentiate the importance of labelled data points by assigning each instance a weight value. For `text/libsvm` input, customers can assign weight values to data instances by attaching them after the labels. For example, `label:weight idx_0:val_0 idx_1:val_1...`. For `text/csv` input, customers need to turn on the `csv_weights` flag in the parameters and attach weight values in the column after labels. For example: `label,weight,val_0,val_1,...`.

## EC2 Instance Recommendation for the XGBoost Algorithm

SageMaker XGBoost 1.0-1 or earlier currently only trains using CPUs. It is a memory-bound (as opposed to compute-bound) algorithm. So, a general-purpose compute instance (for example, M5) is a better choice than a compute-optimized instance (for example, C4). Further, we recommend that you have enough total memory in selected instances to hold the training data. Although it supports the use of disk space to handle data that does not fit into main memory (the out-of-core feature available with the libsvm input mode), writing cache files onto disk slows the algorithm processing time.

SageMaker XGBoost version 1.2 or later supports single-instance GPU training. Despite higher per-instance costs, GPUs train more quickly, making them more cost effective. To take advantage of GPU training, specify the instance type as one of the GPU instances (for example, P3) and set the `tree_method` hyperparameter to `gpu_hist` in your existing XGBoost script. SageMaker currently does not support multi-GPU training.

## XGBoost Sample Notebooks

The following table outlines a variety of sample notebooks that address different use cases of Amazon SageMaker XGBoost algorithm.

Notebook Title	Description
<a href="#">How to Create a Custom XGBoost container?</a>	This notebook shows you how to build a custom XGBoost Container with Amazon SageMaker Batch Transform.
<a href="#">An Introduction to Feature Processing, Training and Inference</a>	This notebook shows you how to build a Machine Learning (ML) Pipeline using Spark Feature Transformers and do real-time inference using Amazon SageMaker Batch Transform.
<a href="#">Regression with XGBoost using Parquet</a>	This notebook shows you how to use the Abalone dataset in Parquet to train a XGBoost model.
<a href="#">How to Train and Host a Multiclass Classification Model?</a>	This notebook shows how to use the MNIST dataset to train and host a multiclass classification model.
<a href="#">How to train a Model for Customer Churn Prediction?</a>	This notebook shows you how to train a model to Predict Mobile Customer Departure in an effort to identify unhappy customers.
<a href="#">An Introduction to Amazon SageMaker Managed Spot infrastructure for XGBoost Training</a>	This notebook shows you how to use Spot Instances for training with a XGBoost Container.
<a href="#">How to use Amazon SageMaker Debugger to debug XGBoost Training Jobs?</a>	This notebook shows you how to use Amazon SageMaker Debugger to monitor training jobs to detect inconsistencies.
<a href="#">How to use Amazon SageMaker Debugger to debug XGBoost Training Jobs in Real-Time?</a>	This notebook shows you how to use the MNIST dataset and Amazon SageMaker Debugger to perform real-time analysis of XGBoost training jobs while training jobs are running.

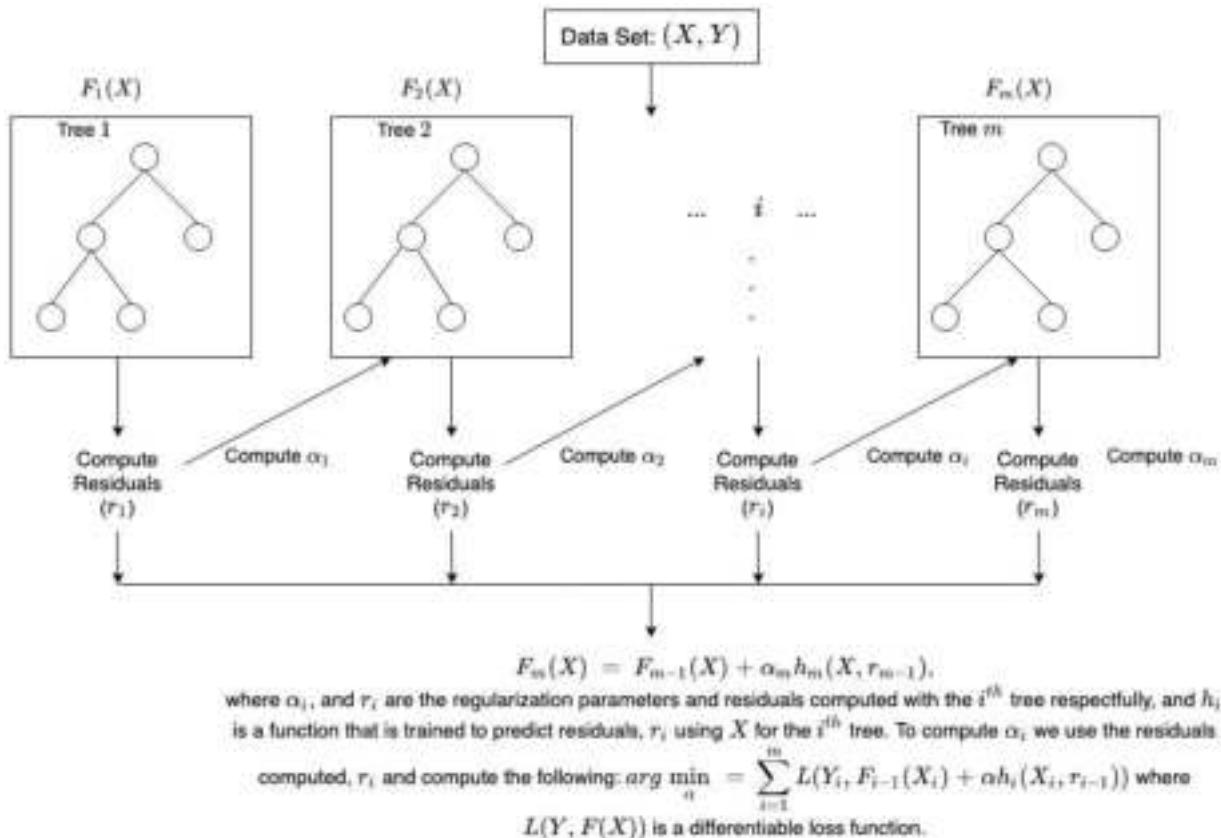
For instructions on how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). After you have created a notebook instance and opened it, choose the **SageMaker Examples** tab to see a list of all of the SageMaker samples. The topic modeling example notebooks using the linear learning algorithm are located in the **Introduction to Amazon algorithms** section. To open a notebook, choose its **Use** tab and choose **Create copy**.

## How XGBoost Works

XGBoost is a popular and efficient open-source implementation of the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm, which attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models.

When using **gradient boosting** for regression, the weak learners are regression trees, and each regression tree maps an input data point to one of its leafs that contains a continuous score. XGBoost minimizes a regularized (L1 and L2) objective function that combines a convex loss function (based on the difference between the predicted and target outputs) and a penalty term for model complexity (in other words, the regression tree functions). The training proceeds iteratively, adding new trees that predict the residuals or errors of prior trees that are then combined with previous trees to make the final prediction. It's called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models.

Below is a brief illustration on how gradient tree boosting works.



For more detail on XGBoost, see:

- [XGBoost: A Scalable Tree Boosting System](#)
- [Gradient Tree Boosting](#)
- [Introduction to Boosted Trees](#)

## XGBoost Hyperparameters

The following table contains the subset of hyperparameters that are required or most commonly used for the Amazon SageMaker XGBoost algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order. The SageMaker XGBoost algorithm is an implementation of the open-source DMLC XGBoost package. Currently SageMaker supports version 1.2-2. For details about full set of hyperparameter that can be configured for this version of XGBoost, see [XGBoost Parameters](#).

Parameter Name	Description
num_class	The number of classes. <b>Required</b> if objective is set to <i>multi:softmax</i> or <i>multi:softprob</i> . Valid values: integer
num_round	The number of rounds to run the training.

Parameter Name	Description
	<b>Required</b> Valid values: integer
alpha	L1 regularization term on weights. Increasing this value makes models more conservative. <b>Optional</b> Valid values: float Default value: 0
base_score	The initial prediction score of all instances, global bias. <b>Optional</b> Valid values: float Default value: 0.5
booster	Which booster to use. The <code>gbtree</code> and <code>dart</code> values use a tree-based model, while <code>gblinear</code> uses a linear function. <b>Optional</b> Valid values: String. One of <code>gbtree</code> , <code>gblinear</code> , or <code>dart</code> . Default value: <code>gbtree</code>
colsample_bylevel	Subsample ratio of columns for each split, in each level. <b>Optional</b> Valid values: Float. Range: [0,1]. Default value: 1
colsample_bynode	Subsample ratio of columns from each node. <b>Optional</b> Valid values: Float. Range: (0,1]. Default value: 1
colsample_bytree	Subsample ratio of columns when constructing each tree. <b>Optional</b> Valid values: Float. Range: [0,1]. Default value: 1

Parameter Name	Description
<code>csv_weights</code>	<p>When this flag is enabled, XGBoost differentiates the importance of instances for csv input by taking the second column (the column after labels) in training data as the instance weights.</p> <p><b>Optional</b></p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>
<code>deterministic_histogram</code>	<p>When this flag is enabled, XGBoost builds histogram on GPU deterministically. Used only if <code>tree_method</code> is set to <code>gpu_hist</code>.</p> <p>For a full list of valid inputs, please refer to <a href="#">XGBoost Parameters</a>.</p> <p><b>Optional</b></p> <p>Valid values: String. Range: <code>true</code> or <code>false</code></p> <p>Default value: <code>true</code></p>
<code>early_stopping_rounds</code>	<p>The model trains until the validation score stops improving. Validation error needs to decrease at least every <code>early_stopping_rounds</code> to continue training. SageMaker hosting uses the best model for inference.</p> <p><b>Optional</b></p> <p>Valid values: integer</p> <p>Default value: -</p>
<code>eta</code>	<p>Step size shrinkage used in updates to prevent overfitting. After each boosting step, you can directly get the weights of new features. The <code>eta</code> parameter actually shrinks the feature weights to make the boosting process more conservative.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: [0,1].</p> <p>Default value: 0.3</p>
<code>eval_metric</code>	<p>Evaluation metrics for validation data. A default metric is assigned according to the objective:</p> <ul style="list-style-type: none"> <li>• <code>rmse</code>: for regression</li> <li>• <code>error</code>: for classification</li> <li>• <code>map</code>: for ranking</li> </ul> <p>For a list of valid inputs, see <a href="#">XGBoost Learning Task Parameters</a>.</p> <p><b>Optional</b></p> <p>Valid values: string</p> <p>Default value: Default according to objective.</p>

Parameter Name	Description
gamma	<p>Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm is.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: <math>[0, \infty)</math>.</p> <p>Default value: 0</p>
grow_policy	<p>Controls the way that new nodes are added to the tree. Currently supported only if <code>tree_method</code> is set to <code>hist</code>.</p> <p><b>Optional</b></p> <p>Valid values: String. Either <code>depthwise</code> or <code>lossguide</code>.</p> <p>Default value: <code>depthwise</code></p>
interaction_constraints	<p>Specify groups of variables that are allowed to interact.</p> <p><b>Optional</b></p> <p>Valid values: Nested list of integers. Each integer represents a feature, and each nested list contains features that are allowed to interact e.g., <code>[[1,2], [3,4,5]]</code>.</p> <p>Default value: None</p>
lambda	<p>L2 regularization term on weights. Increasing this value makes models more conservative.</p> <p><b>Optional</b></p> <p>Valid values: float</p> <p>Default value: 1</p>
lambda_bias	<p>L2 regularization term on bias.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: <math>[0.0, 1.0]</math>.</p> <p>Default value: 0</p>
max_bin	<p>Maximum number of discrete bins to bucket continuous features. Used only if <code>tree_method</code> is set to <code>hist</code>.</p> <p><b>Optional</b></p> <p>Valid values: integer</p> <p>Default value: 256</p>

Parameter Name	Description
max_delta_step	<p>Maximum delta step allowed for each tree's weight estimation. When a positive integer is used, it helps make the update more conservative. The preferred option is to use it in logistic regression. Set it to 1-10 to help control the update.</p> <p><b>Optional</b></p> <p>Valid values: Integer. Range: <math>[0, \infty)</math>.</p> <p>Default value: 0</p>
max_depth	<p>Maximum depth of a tree. Increasing this value makes the model more complex and likely to be overfit. 0 indicates no limit. A limit is required when <code>grow_policy=depth-wise</code>.</p> <p><b>Optional</b></p> <p>Valid values: Integer. Range: <math>[0, \infty)</math></p> <p>Default value: 6</p>
max_leaves	<p>Maximum number of nodes to be added. Relevant only if <code>grow_policy</code> is set to <code>lossguide</code>.</p> <p><b>Optional</b></p> <p>Valid values: integer</p> <p>Default value: 0</p>
min_child_weight	<p>Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than <code>min_child_weight</code>, the building process gives up further partitioning. In linear regression models, this simply corresponds to a minimum number of instances needed in each node. The larger the algorithm, the more conservative it is.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: <math>[0, \infty)</math>.</p> <p>Default value: 1</p>
monotone_constraints	<p>Specifies monotonicity constraints on any feature.</p> <p><b>Optional</b></p> <p>Valid values: Tuple of Integers. Valid integers: -1 (decreasing constraint), 0 (no constraint), 1 (increasing constraint).</p> <p>E.g., (0, 1): No constraint on first predictor, and an increasing constraint on the second. (-1, 1): Decreasing constraint on first predictor, and an increasing constraint on the second.</p> <p>Default value: (0, 0)</p>

Parameter Name	Description
normalize_type	<p>Type of normalization algorithm.</p> <p><b>Optional</b></p> <p>Valid values: Either <i>tree</i> or <i>forest</i>.</p> <p>Default value: <i>tree</i></p>
nthread	<p>Number of parallel threads used to run <i>xgboost</i>.</p> <p><b>Optional</b></p> <p>Valid values: integer</p> <p>Default value: Maximum number of threads.</p>
objective	<p>Specifies the learning task and the corresponding learning objective. Examples: <code>reg:logistic</code>, <code>multi:softmax</code>, <code>reg:squarederror</code>. For a full list of valid inputs, refer to <a href="#">XGBoost Learning Task Parameters</a>.</p> <p><b>Optional</b></p> <p>Valid values: string</p> <p>Default value: <code>reg:squarederror</code></p>
one_drop	<p>When this flag is enabled, at least one tree is always dropped during the dropout.</p> <p><b>Optional</b></p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>
process_type	<p>The type of boosting process to run.</p> <p><b>Optional</b></p> <p>Valid values: String. Either <code>default</code> or <code>update</code>.</p> <p>Default value: <code>default</code></p>
rate_drop	<p>The dropout rate that specifies the fraction of previous trees to drop during the dropout.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: [0.0, 1.0].</p> <p>Default value: 0.0</p>

Parameter Name	Description
<code>refresh_leaf</code>	<p>This is a parameter of the 'refresh' updater plug-in. When set to <code>true</code>(1), tree leaves and tree node stats are updated. When set to <code>false</code>(0), only tree node stats are updated.</p> <p><b>Optional</b></p> <p>Valid values: 0/1</p> <p>Default value: 1</p>
<code>sample_type</code>	<p>Type of sampling algorithm.</p> <p><b>Optional</b></p> <p>Valid values: Either <code>uniform</code> or <code>weighted</code>.</p> <p>Default value: <code>uniform</code></p>
<code>scale_pos_weight</code>	<p>Controls the balance of positive and negative weights. It's useful for unbalanced classes. A typical value to consider: <code>sum(negative cases) / sum(positive cases)</code>.</p> <p><b>Optional</b></p> <p>Valid values: float</p> <p>Default value: 1</p>
<code>seed</code>	<p>Random number seed.</p> <p><b>Optional</b></p> <p>Valid values: integer</p> <p>Default value: 0</p>
<code>single_precision_histogram</code>	<p>When this flag is enabled, XGBoost uses single precision to build histograms instead of double precision. Used only if <code>tree_method</code> is set to <code>hist</code> or <code>gpu_hist</code>.</p> <p>For a full list of valid inputs, please refer to <a href="#">XGBoost Parameters</a>.</p> <p><b>Optional</b></p> <p>Valid values: String. Range: <code>true</code> or <code>false</code></p> <p>Default value: <code>false</code></p>
<code>sketch_eps</code>	<p>Used only for approximate greedy algorithm. This translates into <math>O(1 / \text{sketch\_eps})</math> number of bins. Compared to directly select number of bins, this comes with theoretical guarantee with sketch accuracy.</p> <p><b>Optional</b></p> <p>Valid values: Float, Range: [0, 1].</p> <p>Default value: 0.03</p>

Parameter Name	Description
skip_drop	<p>Probability of skipping the dropout procedure during a boosting iteration.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: [0.0, 1.0].</p> <p>Default value: 0.0</p>
subsample	<p>Subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collects half of the data instances to grow trees. This prevents overfitting.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: [0,1].</p> <p>Default value: 1</p>
tree_method	<p>The tree construction algorithm used in XGBoost.</p> <p><b>Optional</b></p> <p>Valid values: One of <code>auto</code>, <code>exact</code>, <code>approx</code>, <code>hist</code>, or <code>gpu_hist</code>.</p> <p>Default value: <code>auto</code></p>
tweedie_variance_power	<p>Parameter that controls the variance of the Tweedie distribution.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: (1, 2).</p> <p>Default value: 1.5</p>
updater	<p>A comma-separated string that defines the sequence of tree updaters to run. This provides a modular way to construct and to modify the trees.</p> <p>For a full list of valid inputs, please refer to <a href="#">XGBoost Parameters</a>.</p> <p><b>Optional</b></p> <p>Valid values: comma-separated string.</p> <p>Default value: <code>grow_colmaker</code>, <code>prune</code></p>
verbosity	<p>Verbosity of printing messages.</p> <p>Valid values: 0 (silent), 1 (warning), 2 (info), 3 (debug).</p> <p><b>Optional</b></p> <p>Default value: 1</p>

## Tune an XGBoost Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your training and validation datasets. You choose three types of hyperparameters:

- a learning objective function to optimize during model training
- an eval\_metric to use to evaluate model performance during validation
- a set of hyperparameters and a range of values for each to use when tuning the model automatically

You choose the evaluation metric from set of evaluation metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the evaluation metric.

**Note**

Automatic model tuning for XGBoost 0.90 is only available from the Amazon SageMaker SDKs, not from the SageMaker console.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Evaluation Metrics Computed by the XGBoost Algorithm

The XGBoost algorithm computes the following metrics to use for model validation. When tuning the model, choose one of these metrics to evaluate the model. For full list of valid eval\_metric values, refer to [XGBoost Learning Task Parameters](#)

Metric Name	Description	Optimization Direction
validation:accuracy	Classification rate, calculated as #(right)/#(all cases).	Maximize
validation:auc	Area under the curve.	Maximize
validation:error	Binary classification error rate, calculated as #(wrong cases)/#(all cases).	Minimize
validation:f1	Indicator of classification accuracy, calculated as the harmonic mean of precision and recall.	Maximize
validation:logloss	Negative log-likelihood.	Minimize
validation:mae	Mean absolute error.	Minimize
validation:map	Mean average precision.	Maximize
validation:merror	Multiclass classification error rate, calculated as #(wrong cases)/#(all cases).	Minimize
validation:mlogloss	Negative log-likelihood for multiclass classification.	Minimize
validation:mse	Mean squared error.	Minimize
validation:ndcg	Normalized Discounted Cumulative Gain.	Maximize
validation:rmse	Root mean square error.	Minimize

## Tunable XGBoost Hyperparameters

Tune the XGBoost model with the following hyperparameters. The hyperparameters that have the greatest effect on optimizing the XGBoost evaluation metrics are: `alpha`, `min_child_weight`, `subsample`, `eta`, and `num_round`.

Parameter Name	Parameter Type	Recommended Ranges
<code>alpha</code>	ContinuousParameterRanges	MinValue: 0, MaxValue: 1000
<code>colsample_bylevel</code>	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 1
<code>colsample_bynode</code>	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 1
<code>colsample_bytree</code>	ContinuousParameterRanges	MinValue: 0.5, MaxValue: 1
<code>eta</code>	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 0.5
<code>gamma</code>	ContinuousParameterRanges	MinValue: 0, MaxValue: 5
<code>lambda</code>	ContinuousParameterRanges	MinValue: 0, MaxValue: 1000
<code>max_delta_step</code>	IntegerParameterRanges	[0, 10]
<code>max_depth</code>	IntegerParameterRanges	[0, 10]
<code>min_child_weight</code>	ContinuousParameterRanges	MinValue: 0, MaxValue: 120
<code>num_round</code>	IntegerParameterRanges	[1, 4000]
<code>subsample</code>	ContinuousParameterRanges	MinValue: 0.5, MaxValue: 1

## Deprecated Versions of XGBoost and their Upgrades

This topic contains documentation for previous versions of Amazon SageMaker XGBoost that are still available but deprecated. It also provides instructions on how to upgrade deprecated versions of XGBoost, when possible, to more current versions.

### Topics

- [Upgrade XGBoost Version 0.90 to Version 1.2 \(p. 1540\)](#)
- [XGBoost Version 0.72 \(p. 1542\)](#)

### Upgrade XGBoost Version 0.90 to Version 1.2

If you are using the SageMaker Python SDK, to upgrade existing XGBoost 0.90 jobs to version 1.2, you must have version 2.x of the SDK installed and change the `XGBoost version` and `framework_version` parameters to 1.2-2. If you are using Boto3, you need to update the Docker image, and a few hyperparameters and learning objectives.

## Topics

- [Upgrade SageMaker Python SDK Version 1.x to Version 2.x \(p. 1541\)](#)
- [Change the image tag to 1.2-2 \(p. 1541\)](#)
- [Change Docker Image for Boto3 \(p. 1541\)](#)
- [Update Hyperparameters and Learning Objectives \(p. 1542\)](#)

## Upgrade SageMaker Python SDK Version 1.x to Version 2.x

If you are still using Version 1.x of the SageMaker Python SDK, you must upgrade version 2.x of the SageMaker Python SDK. For information on the latest version of the SageMaker Python SDK, see [Use Version 2.x of the SageMaker Python SDK](#). To install the latest version, run:

```
python -m pip install --upgrade sagemaker
```

## Change the image tag to 1.2-2

If you are using the SageMaker Python SDK and using the XGBoost build-in algorithm, change the version parameter in `image_uris.retrieve`.

```
from sagemaker import image_uris
image_uris.retrieve(framework="xgboost", region="us-west-2", version="1.2-2")

estimator = sagemaker.estimator.Estimator(image_uri=xgboost_container,
                                            hyperparameters=hyperparameters,
                                            role=sagemaker.get_execution_role(),
                                            instance_count=1,
                                            instance_type='ml.m5.2xlarge',
                                            volume_size=5, # 5 GB
                                            output_path=output_path)
```

If you are using the SageMaker Python SDK and using XGBoost as a framework to run your customized training scripts, change the `framework_version` parameter in the XGBoost API.

```
estimator = XGBoost(entry_point = "your_xgboost_abalone_script.py",
                     framework_version='1.2-2',
                     hyperparameters=hyperparameters,
                     role=sagemaker.get_execution_role(),
                     instance_count=1,
                     instance_type='ml.m5.2xlarge',
                     output_path=output_path)
```

`sagemaker.session.s3_input` in SageMaker Python SDK version 1.x has been renamed to `sagemaker.inputs.TrainingInput`. You must use `sagemaker.inputs.TrainingInput` as in the following example.

```
content_type = "libsvm"
train_input = TrainingInput("s3://{}//{}//{}//".format(bucket, prefix, 'train'),
                           content_type=content_type)
validation_input = TrainingInput("s3://{}//{}//{}//".format(bucket, prefix, 'validation'),
                                 content_type=content_type)
```

For the full list of SageMaker Python SDK version 2.x changes, see [Use Version 2.x of the SageMaker Python SDK](#).

## Change Docker Image for Boto3

If you are using Boto3 to train or deploy your model, change the docker image tag (0.90-1 or 0.90-2) to 1.2-2.

```
{  
    "AlgorithmSpecification": {  
        "TrainingImage": "746614075791.dkr.ecr.us-west-1.amazonaws.com/sagemaker-xgboost:1.2-2"  
    }  
    ...  
}
```

If you are using the SageMaker Python SDK to retrieve registry path, change the `version` parameter in `image_uris.retrieve`.

```
from sagemaker import image_uris  
image_uris.retrieve(framework="xgboost", region="us-west-2", version="1.2-2")
```

## Update Hyperparameters and Learning Objectives

The `silent` parameter has been deprecated and is no longer available in XGBoost 1.2 and later versions. Use `verbosity` instead. If you were using the `reg:linear` learning objective, it has been deprecated as well in favor of `reg:squarederror`. Use `reg:squarederror` instead.

```
hyperparameters = {  
    "verbosity": "2",  
    "objective": "reg:squarederror",  
    "num_round": "50",  
    ...  
}  
  
estimator = sagemaker.estimator.Estimator(image_uri=xgboost_container,  
                                         hyperparameters=hyperparameters,  
                                         ...)
```

## XGBoost Version 0.72

### Important

The XGBoost 0.72 is deprecated by Amazon SageMaker. You can still use this old version of XGBoost (as a built-in algorithm) by pulling its image URI as shown in the following code sample. For XGBoost, the image URI ending with `:1` is for the old version.

### SageMaker Python SDK v1

```
import boto3  
from sagemaker.amazon.amazon_estimator import get_image_uri  
  
xgb_image_uri = get_image_uri(boto3.Session().region_name, "xgboost",  
                           repo_version="1")
```

### SageMaker Python SDK v2

```
import boto3  
from sagemaker import image_uris  
  
xgb_image_uri = image_uris.retrieve("xgboost", boto3.Session().region_name, "1")
```

If you want to use newer versions, you have to explicitly specify the image URI tags (see [Supported versions \(p. 1525\)](#)).

This previous release of the Amazon SageMaker XGBoost algorithm is based on the 0.72 release. **XGBoost** (eXtreme Gradient Boosting) is a popular and efficient open-source implementation of

the gradient boosted trees algorithm. Gradient boosting is a supervised learning algorithm that attempts to accurately predict a target variable by combining the estimates of a set of simpler, weaker models. XGBoost has done remarkably well in machine learning competitions because it robustly handles a variety of data types, relationships, and distributions, and because of the large number of hyperparameters that can be tweaked and tuned for improved fits. This flexibility makes XGBoost a solid choice for problems in regression, classification (binary and multiclass), and ranking.

Customers should consider using the new release of [XGBoost Algorithm \(p. 1524\)](#). They can use it as a SageMaker built-in algorithm or as a framework to run scripts in their local environments as they would typically, for example, do with a Tensorflow deep learning framework. The new implementation has a smaller memory footprint, better logging, improved hyperparameter validation, and an expanded set of metrics. The earlier implementation of XGBoost remains available to customers if they need to postpone migrating to the new version. But this previous implementation will remain tied to the 0.72 release of XGBoost.

### [Input/Output Interface for the XGBoost Release 0.72](#)

Gradient boosting operates on tabular data, with the rows representing observations, one column representing the target variable or label, and the remaining columns representing features.

The SageMaker implementation of XGBoost supports CSV and libsvm formats for training and inference:

- For Training ContentType, valid inputs are *text/libsvm* (default) or *text/csv*.
- For Inference ContentType, valid inputs are *text/libsvm* or (the default) *text/csv*.

#### **Note**

For CSV training, the algorithm assumes that the target variable is in the first column and that the CSV does not have a header record. For CSV inference, the algorithm assumes that CSV input does not have the label column.

For libsvm training, the algorithm assumes that the label is in the first column. Subsequent columns contain the zero-based index value pairs for features. So each row has the format: <label> <index0>:<value0> <index1>:<value1> ... Inference requests for libsvm may or may not have labels in the libsvm format.

This differs from other SageMaker algorithms, which use the protobuf training input format to maintain greater consistency with standard XGBoost data formats.

For CSV training input mode, the total memory available to the algorithm (Instance Count \* the memory available in the `InstanceType`) must be able to hold the training dataset. For libsvm training input mode, it's not required, but we recommend it.

SageMaker XGBoost uses the Python pickle module to serialize/deserialize the model, which can be used for saving/loading the model.

### **To use a model trained with SageMaker XGBoost in open source XGBoost**

- Use the following Python code:

```
import pickle as pkl
import tarfile
import xgboost

t = tarfile.open('model.tar.gz', 'r:gz')
t.extractall()

model = pkl.load(open(model_file_path, 'rb'))

# prediction with test data
```

```
pred = model.predict(dttest)
```

## To differentiate the importance of labelled data points use Instance Weight Supports

- SageMaker XGBoost allows customers to differentiate the importance of labelled data points by assigning each instance a weight value. For `text/libsvm` input, customers can assign weight values to data instances by attaching them after the labels. For example, `label:weight idx_0:val_0 idx_1:val_1....` For `text/csv` input, customers need to turn on the `csv_weights` flag in the parameters and attach weight values in the column after labels. For example: `label,weight,val_0,val_1,...`.

## EC2 Instance Recommendation for the XGBoost Release 0.72

SageMaker XGBoost currently only trains using CPUs. It is a memory-bound (as opposed to compute-bound) algorithm. So, a general-purpose compute instance (for example, M4) is a better choice than a compute-optimized instance (for example, C4). Further, we recommend that you have enough total memory in selected instances to hold the training data. Although it supports the use of disk space to handle data that does not fit into main memory (the out-of-core feature available with the `libsvm` input mode), writing cache files onto disk slows the algorithm processing time.

### XGBoost Release 0.72 Sample Notebooks

For a sample notebook that shows how to use the latest version of SageMaker XGBoost as a built-in algorithm to train and host a regression model, see [Regression with Amazon SageMaker XGBoost algorithm](#). To use the 0.72 version of XGBoost, you need to change the version in the sample code to 0.72. For instructions how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the SageMaker samples. The topic modeling example notebooks using the XGBoost algorithms are located in the **Introduction to Amazon algorithms** section. To open a notebook, click on its **Use** tab and select **Create copy**.

### XGBoost Release 0.72 Hyperparameters

The following table contains the hyperparameters for the XGBoost algorithm. These are parameters that are set by users to facilitate the estimation of model parameters from data. The required hyperparameters that must be set are listed first, in alphabetical order. The optional hyperparameters that can be set are listed next, also in alphabetical order. The SageMaker XGBoost algorithm is an implementation of the open-source XGBoost package. Currently SageMaker supports version 0.72. For more detail about hyperparameter configuration for this version of XGBoost, see [XGBoost Parameters](#).

Parameter Name	Description
<code>num_class</code>	The number of classes. <b>Required</b> if <code>objective</code> is set to <code>multi:softmax</code> or <code>multi:softprob</code> . Valid values: integer
<code>num_round</code>	The number of rounds to run the training. <b>Required</b> Valid values: integer
<code>alpha</code>	L1 regularization term on weights. Increasing this value makes models more conservative.

Parameter Name	Description
	<b>Optional</b> Valid values: float Default value: 0
<code>base_score</code>	The initial prediction score of all instances, global bias. <b>Optional</b> Valid values: float Default value: 0.5
<code>booster</code>	Which booster to use. The <code>gbtree</code> and <code>dart</code> values use a tree-based model, while <code>gblinear</code> uses a linear function. <b>Optional</b> Valid values: String. One of <code>gbtree</code> , <code>gblinear</code> , or <code>dart</code> . Default value: <code>gbtree</code>
<code>colsample_bylevel</code>	Subsample ratio of columns for each split, in each level. <b>Optional</b> Valid values: Float. Range: [0,1]. Default value: 1
<code>colsample_bytree</code>	Subsample ratio of columns when constructing each tree. <b>Optional</b> Valid values: Float. Range: [0,1]. Default value: 1
<code>csv_weights</code>	When this flag is enabled, XGBoost differentiates the importance of instances for csv input by taking the second column (the column after labels) in training data as the instance weights. <b>Optional</b> Valid values: 0 or 1 Default value: 0
<code>early_stopping_rounds</code>	The model trains until the validation score stops improving. Validation error needs to decrease at least every <code>early_stopping_rounds</code> to continue training. SageMaker hosting uses the best model for inference. <b>Optional</b> Valid values: integer Default value: -

Parameter Name	Description
eta	<p>Step size shrinkage used in updates to prevent overfitting. After each boosting step, you can directly get the weights of new features. The eta parameter actually shrinks the feature weights to make the boosting process more conservative.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: [0,1].</p> <p>Default value: 0.3</p>
eval_metric	<p>Evaluation metrics for validation data. A default metric is assigned according to the objective:</p> <ul style="list-style-type: none"> <li>• <code>rmse</code>: for regression</li> <li>• <code>error</code>: for classification</li> <li>• <code>map</code>: for ranking</li> </ul> <p>For a list of valid inputs, see <a href="#">XGBoost Parameters</a>.</p> <p><b>Optional</b></p> <p>Valid values: string</p> <p>Default value: Default according to objective.</p>
gamma	<p>Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger, the more conservative the algorithm is.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: [0,∞).</p> <p>Default value: 0</p>
grow_policy	<p>Controls the way that new nodes are added to the tree. Currently supported only if <code>tree_method</code> is set to <code>hist</code>.</p> <p><b>Optional</b></p> <p>Valid values: String. Either <code>depthwise</code> or <code>lossguide</code>.</p> <p>Default value: <code>depthwise</code></p>
lambda	<p>L2 regularization term on weights. Increasing this value makes models more conservative.</p> <p><b>Optional</b></p> <p>Valid values: float</p> <p>Default value: 1</p>

Parameter Name	Description
<code>lambda_bias</code>	<p>L2 regularization term on bias.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: [0.0, 1.0].</p> <p>Default value: 0</p>
<code>max_bin</code>	<p>Maximum number of discrete bins to bucket continuous features. Used only if <code>tree_method</code> is set to <code>hist</code>.</p> <p><b>Optional</b></p> <p>Valid values: integer</p> <p>Default value: 256</p>
<code>max_delta_step</code>	<p>Maximum delta step allowed for each tree's weight estimation. When a positive integer is used, it helps make the update more conservative. The preferred option is to use it in logistic regression. Set it to 1-10 to help control the update.</p> <p><b>Optional</b></p> <p>Valid values: Integer. Range: [0,∞).</p> <p>Default value: 0</p>
<code>max_depth</code>	<p>Maximum depth of a tree. Increasing this value makes the model more complex and likely to be overfit. 0 indicates no limit. A limit is required when <code>grow_policy=depth-wise</code>.</p> <p><b>Optional</b></p> <p>Valid values: Integer. Range: [0,∞)</p> <p>Default value: 6</p>
<code>max_leaves</code>	<p>Maximum number of nodes to be added. Relevant only if <code>grow_policy</code> is set to <code>lossguide</code>.</p> <p><b>Optional</b></p> <p>Valid values: integer</p> <p>Default value: 0</p>
<code>min_child_weight</code>	<p>Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than <code>min_child_weight</code>, the building process gives up further partitioning. In linear regression models, this simply corresponds to a minimum number of instances needed in each node. The larger the algorithm, the more conservative it is.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: [0,∞).</p> <p>Default value: 1</p>

Parameter Name	Description
normalize_type	<p>Type of normalization algorithm.</p> <p><b>Optional</b></p> <p>Valid values: Either <i>tree</i> or <i>forest</i>.</p> <p>Default value: <i>tree</i></p>
nthread	<p>Number of parallel threads used to run <i>xgboost</i>.</p> <p><b>Optional</b></p> <p>Valid values: integer</p> <p>Default value: Maximum number of threads.</p>
objective	<p>Specifies the learning task and the corresponding learning objective. Examples: <i>reg:logistic</i>, <i>reg:softmax</i>, <i>multi:squarederror</i>. For a full list of valid inputs, refer to <a href="#">XGBoost Parameters</a>.</p> <p><b>Optional</b></p> <p>Valid values: string</p> <p>Default value: <i>reg:squarederror</i></p>
one_drop	<p>When this flag is enabled, at least one tree is always dropped during the dropout.</p> <p><b>Optional</b></p> <p>Valid values: 0 or 1</p> <p>Default value: 0</p>
process_type	<p>The type of boosting process to run.</p> <p><b>Optional</b></p> <p>Valid values: String. Either <i>default</i> or <i>update</i>.</p> <p>Default value: <i>default</i></p>
rate_drop	<p>The dropout rate that specifies the fraction of previous trees to drop during the dropout.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: [0.0, 1.0].</p> <p>Default value: 0.0</p>

Parameter Name	Description
<code>refresh_leaf</code>	<p>This is a parameter of the 'refresh' updater plug-in. When set to <code>true</code> (1), tree leaves and tree node stats are updated. When set to <code>false</code>(0), only tree node stats are updated.</p> <p><b>Optional</b></p> <p>Valid values: 0/1</p> <p>Default value: 1</p>
<code>sample_type</code>	<p>Type of sampling algorithm.</p> <p><b>Optional</b></p> <p>Valid values: Either <code>uniform</code> or <code>weighted</code>.</p> <p>Default value: <code>uniform</code></p>
<code>scale_pos_weight</code>	<p>Controls the balance of positive and negative weights. It's useful for unbalanced classes. A typical value to consider: <code>sum(negative cases) / sum(positive cases)</code>.</p> <p><b>Optional</b></p> <p>Valid values: float</p> <p>Default value: 1</p>
<code>seed</code>	<p>Random number seed.</p> <p><b>Optional</b></p> <p>Valid values: integer</p> <p>Default value: 0</p>
<code>silent</code>	<p>0 means print running messages, 1 means silent mode.</p> <p>Valid values: 0 or 1</p> <p><b>Optional</b></p> <p>Default value: 0</p>
<code>sketch_eps</code>	<p>Used only for approximate greedy algorithm. This translates into <math>O(1 / \text{sketch\_eps})</math> number of bins. Compared to directly select number of bins, this comes with theoretical guarantee with sketch accuracy.</p> <p><b>Optional</b></p> <p>Valid values: Float, Range: [0, 1].</p> <p>Default value: 0.03</p>

Parameter Name	Description
<code>skip_drop</code>	<p>Probability of skipping the dropout procedure during a boosting iteration.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: [0.0, 1.0].</p> <p>Default value: 0.0</p>
<code>subsample</code>	<p>Subsample ratio of the training instance. Setting it to 0.5 means that XGBoost randomly collects half of the data instances to grow trees. This prevents overfitting.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: [0,1].</p> <p>Default value: 1</p>
<code>tree_method</code>	<p>The tree construction algorithm used in XGBoost.</p> <p><b>Optional</b></p> <p>Valid values: One of <code>auto</code>, <code>exact</code>, <code>approx</code>, or <code>hist</code>.</p> <p>Default value: <code>auto</code></p>
<code>tweedie_variance_power</code>	<p>Parameter that controls the variance of the Tweedie distribution.</p> <p><b>Optional</b></p> <p>Valid values: Float. Range: (1, 2).</p> <p>Default value: 1.5</p>
<code>updater</code>	<p>A comma-separated string that defines the sequence of tree updaters to run. This provides a modular way to construct and to modify the trees.</p> <p>For a full list of valid inputs, please refer to <a href="#">XGBoost Parameters</a>.</p> <p><b>Optional</b></p> <p>Valid values: comma-separated string.</p> <p>Default value: <code>grow_colmaker</code>, <code>prune</code></p>

## Tune an XGBoost Release 0.72 Model

*Automatic model tuning*, also known as hyperparameter tuning, finds the best version of a model by running many jobs that test a range of hyperparameters on your training and validation datasets. You choose three types of hyperparameters:

- a learning objective function to optimize during model training
- an `eval_metric` to use to evaluate model performance during validation
- a set of hyperparameters and a range of values for each to use when tuning the model automatically

You choose the evaluation metric from set of evaluation metrics that the algorithm computes. Automatic model tuning searches the hyperparameters chosen to find the combination of values that result in the model that optimizes the evaluation metric.

For more information about model tuning, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

### Metrics Computed by the XGBoost Release 0.72 Algorithm

The XGBoost algorithm based on version 0.72 computes the following nine metrics to use for model validation. When tuning the model, choose one of these metrics to evaluate the model. For full list of valid `eval_metric` values, refer to [XGBoost Learning Task Parameters](#)

Metric Name	Description	Optimization Direction
<code>validation:auc</code>	Area under the curve.	Maximize
<code>validation:error</code>	Binary classification error rate, calculated as #(wrong cases)/#(all cases).	Minimize
<code>validation:logloss</code>	Negative log-likelihood.	Minimize
<code>validation:mae</code>	Mean absolute error.	Minimize
<code>validation:map</code>	Mean average precision.	Maximize
<code>validation:merror</code>	Multiclass classification error rate, calculated as #(wrong cases)/#(all cases).	Minimize
<code>validation:mlogloss</code>	Negative log-likelihood for multiclass classification.	Minimize
<code>validation:ndcg</code>	Normalized Discounted Cumulative Gain.	Maximize
<code>validation:rmse</code>	Root mean square error.	Minimize

### Tunable XGBoost Release 0.72 Hyperparameters

Tune the XGBoost model with the following hyperparameters. The hyperparameters that have the greatest effect on optimizing the XGBoost evaluation metrics are: `alpha`, `min_child_weight`, `subsample`, `eta`, and `num_round`.

Parameter Name	Parameter Type	Recommended Ranges
<code>alpha</code>	ContinuousParameterRanges	MinValue: 0, MaxValue: 1000
<code>colsample_bylevel</code>	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 1
<code>colsample_bytree</code>	ContinuousParameterRanges	MinValue: 0.5, MaxValue: 1
<code>eta</code>	ContinuousParameterRanges	MinValue: 0.1, MaxValue: 0.5
<code>gamma</code>	ContinuousParameterRanges	MinValue: 0, MaxValue: 5

Parameter Name	Parameter Type	Recommended Ranges
lambda	ContinuousParameterRanges	MinValue: 0, MaxValue: 1000
max_delta_step	IntegerParameterRanges	[0, 10]
max_depth	IntegerParameterRanges	[0, 10]
min_child_weight	ContinuousParameterRanges	MinValue: 0, MaxValue: 120
num_round	IntegerParameterRanges	[1, 4000]
subsample	ContinuousParameterRanges	MinValue: 0.5, MaxValue: 1

## Use Reinforcement Learning with Amazon SageMaker

Reinforcement learning (RL) combines fields such as computer science, neuroscience, and psychology to determine how to map situations to actions to maximize a numerical reward signal. This notion of a reward signal in RL stems from neuroscience research into how the human brain makes decisions about which actions maximize reward and minimize punishment. In most situations, humans are not given explicit instructions on which actions to take, but instead must learn both which actions yield the most immediate rewards, and how those actions influence future situations and consequences.

The problem of RL is formalized using Markov decision processes (MDPs) that originate from dynamical systems theory. MDPs aim to capture high-level details of a real problem that a learning agent encounters over some period of time in attempting to achieve some ultimate goal. The learning agent should be able to determine the current state of its environment and identify possible actions that affect the learning agent's current state. Furthermore, the learning agent's goals should correlate strongly to the state of the environment. A solution to a problem formulated in this way is known as a reinforcement learning method.

### What are the differences between reinforcement, supervised, and unsupervised learning paradigms?

Machine learning can be divided into three distinct learning paradigms: supervised, unsupervised, and reinforcement.

In supervised learning, an external supervisor provides a training set of labeled examples. Each example contains information about a situation, belongs to a category, and has a label identifying the category to which it belongs. The goal of supervised learning is to generalize in order to predict correctly in situations that are not present in the training data.

In contrast, RL deals with interactive problems, making it infeasible to gather all possible examples of situations with correct labels that an agent might encounter. This type of learning is most promising when an agent is able to accurately learn from its own experience and adjust accordingly.

In unsupervised learning, an agent learns by uncovering structure within unlabeled data. While a RL agent might benefit from uncovering structure based on its experiences, the sole purpose of RL is to maximize a reward signal.

# Manage Machine Learning with Amazon SageMaker Experiments

Amazon SageMaker Experiments is a capability of Amazon SageMaker that lets you organize, track, compare, and evaluate your machine learning experiments.

Machine learning is an iterative process. You need to experiment with multiple combinations of data, algorithm and parameters, all the while observing the impact of incremental changes on model accuracy. Over time this iterative experimentation can result in thousands of model training runs and model versions. This makes it hard to track the best performing models and their input configurations. It's also difficult to compare active experiments with past experiments to identify opportunities for further incremental improvements.

SageMaker Experiments automatically tracks the inputs, parameters, configurations, and results of your iterations as *trials*. You can assign, group, and organize these trials into *experiments*. SageMaker Experiments is integrated with Amazon SageMaker Studio providing a visual interface to browse your active and past experiments, compare trials on key performance metrics, and identify the best performing models.

SageMaker Experiments comes with its own [Experiments Python SDK](#) which makes the analytics capabilities easily accessible in Amazon SageMaker Notebooks. Because SageMaker Experiments enables tracking of all the steps and artifacts that went into creating a model, you can quickly revisit the origins of a model when you are troubleshooting issues in production, or auditing your models for compliance verifications.

## Topics

- [SageMaker Experiments Features \(p. 1553\)](#)
- [Create an Amazon SageMaker Experiment \(p. 1554\)](#)
- [View and Compare Amazon SageMaker Experiments, Trials, and Trial Components \(p. 1558\)](#)
- [Track and Compare Tutorial \(p. 1561\)](#)
- [Search Experiments Using Amazon SageMaker Studio \(p. 1567\)](#)
- [Clean Up Amazon SageMaker Experiment Resources \(p. 1572\)](#)
- [Search Using the Amazon SageMaker Console and API \(p. 1574\)](#)

## SageMaker Experiments Features

The following sections provide a brief overview of the features provided by SageMaker Experiments.

## Topics

- [Organize Experiments \(p. 1553\)](#)
- [Track Experiments \(p. 1554\)](#)
- [Compare and Evaluate Experiments \(p. 1554\)](#)
- [Amazon SageMaker Autopilot \(p. 1554\)](#)

## Organize Experiments

Amazon SageMaker Experiments offers a structured organization scheme to help users group and organize their machine learning iterations. The top level entity, an *experiment*, is a collection of *trials* that are observed, compared, and evaluated as a group. A trial is a set of steps called *trial components*.

Each trial component can include a combination of inputs such as datasets, algorithms, and parameters, and produce specific outputs such as models, metrics, datasets, and checkpoints. Examples of trial components are data pre-processing jobs, training jobs, and batch transform jobs.

The goal of an experiment is to determine the trial that produces the best model. Multiple trials are performed, each one isolating and measuring the impact of a change to one or more inputs, while keeping the remaining inputs constant. By analyzing the trials, you can determine which features have the most effect on the model.

## Track Experiments

Amazon SageMaker Experiments enables tracking of experiments.

### Automated Tracking

SageMaker Experiments automatically tracks Amazon SageMaker Autopilot jobs as experiments with their underlying training jobs tracked as trials. SageMaker Experiments also automatically tracks SageMaker independently executed training, batch transform, and processing jobs as trial components, whether assigned to a trial or left unassigned. Unassigned trial components can be associated with a trial at a later time. All experiment artifacts including datasets, algorithms, hyperparameters, and model metrics are tracked and recorded. This data allows customers to trace the complete lineage of a model which helps with model governance, auditing, and compliance verifications.

### Manual Tracking

SageMaker Experiments provides [tracking APIs](#) for recording and tracking machine learning workflows running locally on SageMaker Studio notebooks, including classic SageMaker notebooks. These experiments must be part of a SageMaker training, batch transform, or processing job.

## Compare and Evaluate Experiments

Amazon SageMaker Experiments is integrated with Amazon SageMaker Studio. When you use SageMaker Studio, SageMaker Experiments automatically tracks your experiments and trials, and presents visualizations of the tracked data and an interface to search the data.

SageMaker Experiments automatically organizes, ranks, and sorts trials based on a chosen metric using the concept of a trial leaderboard. SageMaker Studio produces real-time data visualizations, such as metric charts and graphs, to quickly compare and identify the best performing models. These are updated in real-time as the experiment progresses.

## Amazon SageMaker Autopilot

Amazon SageMaker Experiments is integrated with Amazon SageMaker Autopilot. When you perform an Autopilot job, SageMaker Experiments creates an experiment for the job, and trials for each of the different combinations of the available trial components, parameters, and artifacts. You can visually drill into all trials and components using SageMaker Studio.

## Create an Amazon SageMaker Experiment

Create an Amazon SageMaker experiment to track your SageMaker training, processing, and transform jobs.

The following procedure shows you how to create a SageMaker experiment for a SageMaker training, processing, or transform job. Steps labeled as (Studio) describe how to view the experiment in Amazon SageMaker Studio. You don't have to run the experiment in Studio to view the experiment in Studio.

For a tutorial that shows this functionality in an existing SageMaker Studio notebook, see [Track and Compare Tutorial \(p. 1561\)](#).

1. Import the `sys` module to install the SDKs.

```
import sys
```

2. (Optional) The [Amazon SageMaker Python SDK](#), comes preinstalled in SageMaker Studio. If you plan to run your code outside Studio, install the SageMaker Python SDK.

```
!{sys.executable} -m pip install sagemaker
```

3. Install the [SageMaker Experiments Python SDK](#).

```
!{sys.executable} -m pip install sagemaker-experiments
```

4. Import modules.

```
import time
from time import strftime

import sagemaker

from smexperiments.experiment import Experiment
from smexperiments.trial import Trial
from smexperiments.trial_component import TrialComponent
from smexperiments.tracker import Tracker
```

5. Get the execution role and create the SageMaker session.

```
role = sagemaker.get_execution_role()
sm_sess = sagemaker.Session()
```

6. Create a SageMaker experiment. The experiment name must be unique in your account.

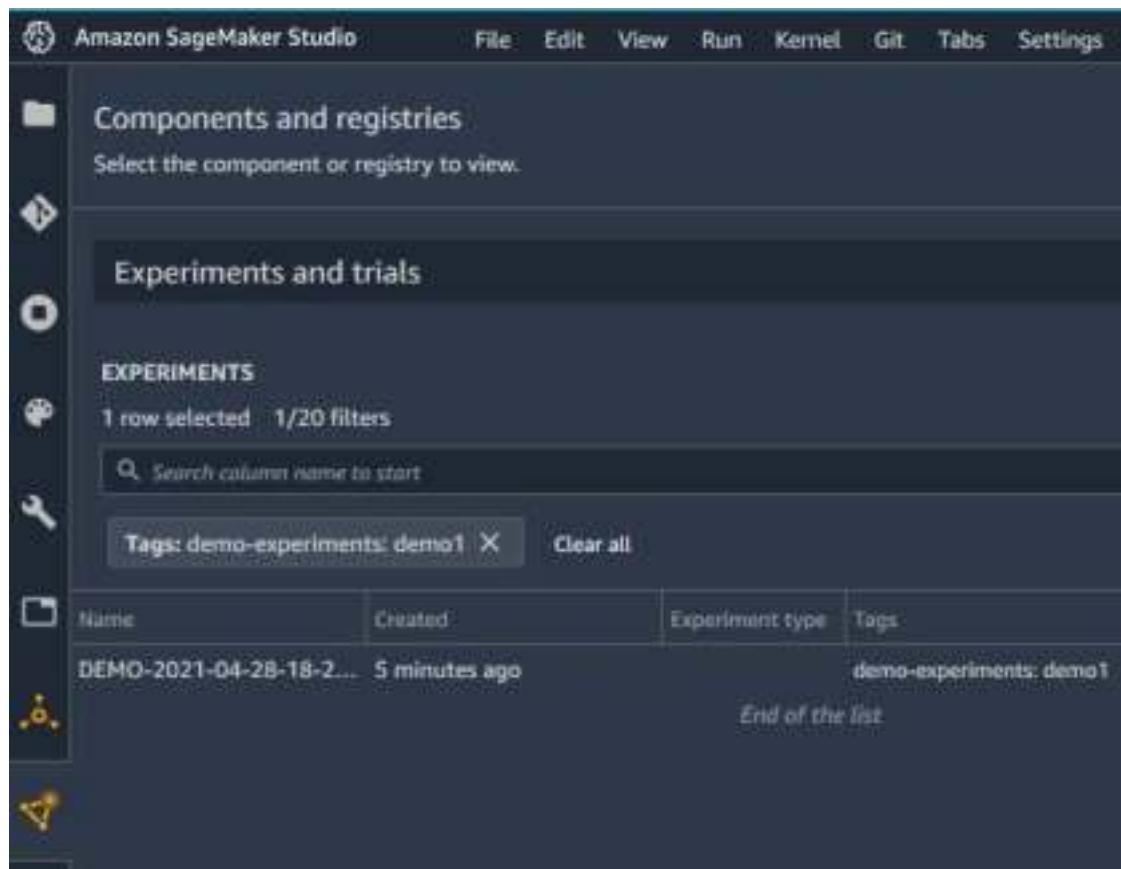
**Note**

The `tags` parameter is optional. You can search for the tag using Studio, the SageMaker console, and the SDK. Tags can also be applied to trials and trial components. For information on how to search tags using Studio, see [Search by Tag \(p. 1571\)](#).

```
create_date = strftime("%Y-%m-%d-%H-%M-%S")
demo_experiment = Experiment.create(experiment_name = "DEMO-{}".format(create_date),
                                     description = "Demo experiment",
                                     tags = [{'Key': 'demo-experiments', 'Value': 'demo1'}])
```

7. (Studio) To view the experiment in SageMaker Studio, in the left sidebar, choose the **Components and registries** icon (). In the drop-down menu, select **Experiments and trials** to display the experiments browser.

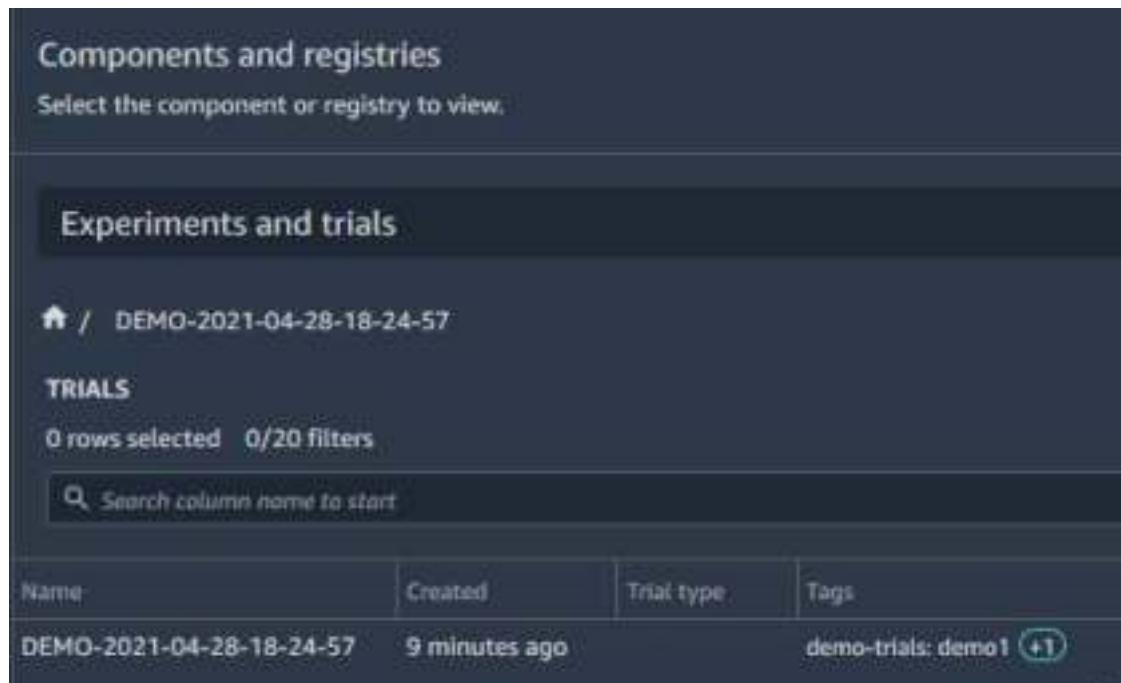
After the code runs, the experiment list contains the new experiment. It might take a moment for the list to refresh and display the experiment. The filter on the experiment tag is also displayed. Only experiments that have a matching tag are displayed. Your list should look similar to the following:



8. Create a trial for the experiment. The trial name must be unique in your account.

```
demo_trial = Trial.create(trial_name = "DEMO-{}".format(create_date),
                           experiment_name = demo_experiment.experiment_name,
                           tags = [{"Key": 'demo-trials', 'Value': 'demo1'}])
```

9. (Studio) In the experiment list, double-click the experiment to display a list of the trials in the experiment (this example has one trial). Your list should look similar to the following:



10. Create a trial component as part of the trial. The trial component is the SageMaker job.

Add the [ExperimentConfig](#) parameter to the appropriate method. The SageMaker jobs listed in the following table are supported.

Job	SageMaker Python SDK method	Boto3 method
Training	<a href="#">Estimator.fit</a>	<a href="#">CreateTrainingJob</a>
Processing	<a href="#">Processor.run</a>	<a href="#">CreateProcessingJob</a>
Transform	<a href="#">Transformer.transform</a>	<a href="#">CreateTransformJob</a>

The following examples are for a training job. The `Tags` parameter adds a tag to the trial component. `ExperimentName` isn't specified because the trial was associated with the experiment when the trial was created in an earlier step.

### Using the SageMaker Python SDK

```
sagemaker.estimator.Estimator(  
    ...,  
    sagemaker_session = sm_sess,  
    tags = [{'Key': 'demo-jobs', 'Value': 'demo2'}])  
  
estimator.fit(  
    ...,  
    experiment_config = {  
        # "ExperimentName"  
        "TrialName" : demo_trial.trial_name,  
        "TrialComponentDisplayName" : "TrainingJob",  
    })
```

### Using Boto3

```
create_training_job(  
    ...  
    "ExperimentConfig": {  
        # "ExperimentName"  
        "TrialName" : demo_trial.trial_name,  
        "TrialComponentDisplayName" : "TrainingJob",  
    },  
    "Tags": [{"Key': 'demo-jobs', 'Value': 'demo2'}])
```

11. (Studio) In the trial list, double-click the trial to display a list of the components in the trial (this example has one trial). Your list should look similar to the following:

The screenshot shows a dark-themed user interface for Amazon SageMaker Studio. At the top, there is a breadcrumb navigation bar with icons for Home and a trial identifier: / DEMO-2020-07-15-16-36-08 / DEMO-2020-07-15-16-36-08 /. Below this is a section titled "TRIAL COMPONENTS" with the subtext "0 rows selected 0/20 filters". A search bar contains the placeholder "Search column name to start". A "Clear all" button is present. The main area displays a table with four columns: "Name", "Created", "Trial component type", and "Tags". One row is visible, showing "TrainingJob" under "Name", "9 minutes ago" under "Created", "Training job" under "Trial component type", and "demo-jobs: demo2" under "Tags". A message "End of the list" is at the bottom right of the table area.

Name	Created	Trial component type	Tags
TrainingJob	9 minutes ago	Training job	demo-jobs: demo2

12. (Studio) To view information about the experiment, trial, and job (trial component), see [View and Compare Amazon SageMaker Experiments, Trials, and Trial Components \(p. 1558\)](#).

To clean up the resources you created, see [Clean Up Amazon SageMaker Experiment Resources \(p. 1572\)](#).

## View and Compare Amazon SageMaker Experiments, Trials, and Trial Components

An Amazon SageMaker *experiment* consists of multiple *trials* with a related objective. A trial consists of one or more *trial components*, such as a data preprocessing job and a training job.

You use the experiments browser to display a list of these entities. You can filter the list by entity name, type, and tags. The entities are presented in a hierarchical view (that is, experiments > trials > trial components). Double-click an entity to see other entities that are at a lower level in the hierarchy. Use the breadcrumbs above the list to move to a higher level in the hierarchy.

For a tutorial using a SageMaker example notebook, see [Track and Compare Tutorial \(p. 1561\)](#). For an overview of the Studio user interface, see [Amazon SageMaker Studio UI Overview \(p. 69\)](#).

### Topics

- [View Experiments, Trials, and Trial Components \(p. 1559\)](#)
- [Compare Experiments, Trials, and Trial Components \(p. 1560\)](#)

## View Experiments, Trials, and Trial Components

Amazon SageMaker Studio provides an experiments browser that you can use to view lists of experiments, trials, and trial components. You can choose one of these entities to view detailed information about the entity or choose multiple entities for comparison.

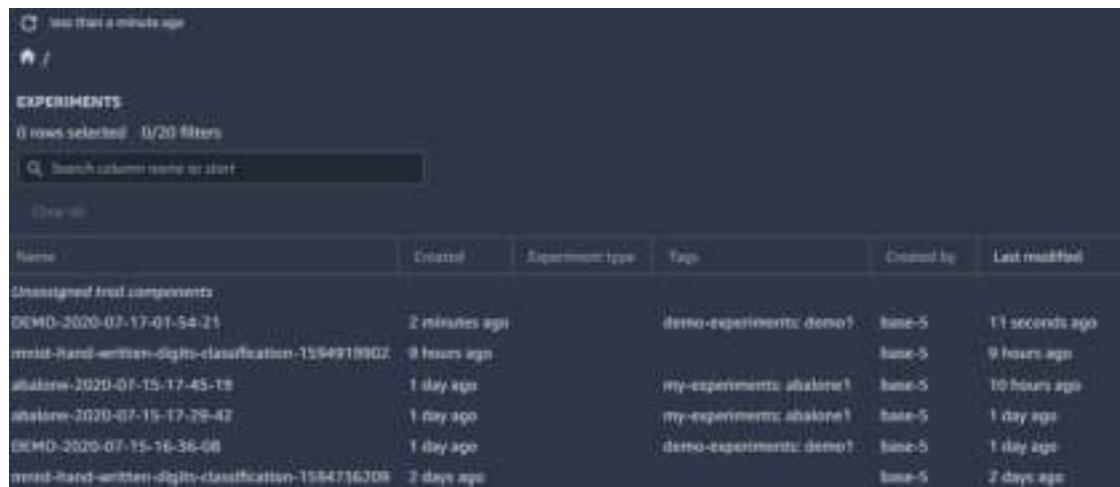
### To view experiments, trials, and trial components

1. In the left sidebar of Studio, choose the **Components and registries** icon (💡). In the drop-down menu, select **Experiments and trials** to display the experiments browser.

A list of experiments and their properties is displayed. The list includes all the SageMaker experiments in your account, including experiments created outside of SageMaker Studio.

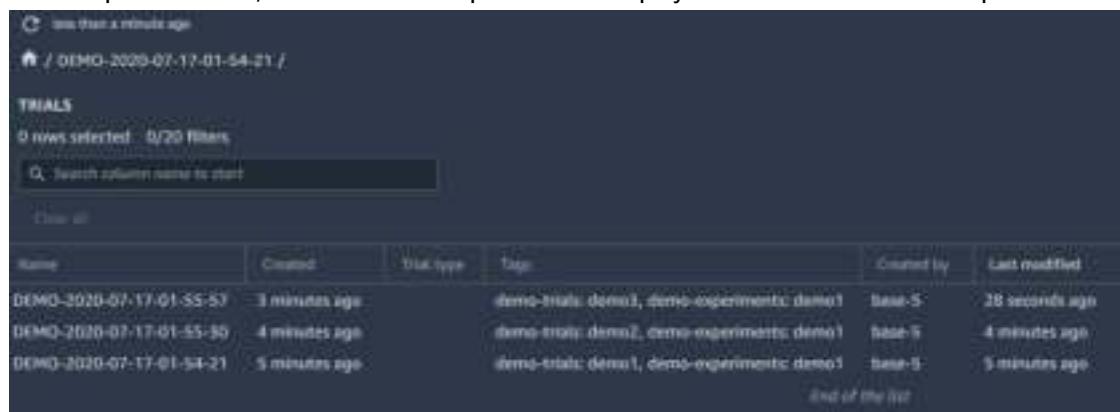
#### Note

To view all the properties, you might have to expand the width of the experiments browser by dragging the right border.



Name	Created	Experiment type	Tags	Created by	Last modified
Untagged trial components					
DEMO-2020-07-17-01-54-21	2 minutes ago		demo-experiments: demo1	base-S	11 seconds ago
mnist-hand-written-digits-classification-1584919902	9 hours ago			base-S	9 hours ago
analone-2020-07-15-17-45-13	1 day ago		my-experiments: analone1	base-S	10 hours ago
analone-2020-07-15-17-29-42	1 day ago		my-experiments: analone1	base-S	1 day ago
DEMO-2020-07-15-16-36-08	1 day ago		demo-experiments: demo1	base-S	1 day ago
mnist-hand-written-digits-classification-1584796308	2 days ago			base-S	2 days ago

2. In the experiments list, double-click an experiment to display a list of the trials in the experiment.



Name	Created	Trial type	Tags	Created by	Last modified
DEMO-2020-07-17-01-54-21 /					
DEMO-2020-07-17-01-55-53	3 minutes ago		demo-trials: demo1, demo-experiments: demo1	base-S	38 seconds ago
DEMO-2020-07-17-01-55-30	4 minutes ago		demo-trials: demo2, demo-experiments: demo1	base-S	4 minutes ago
DEMO-2020-07-17-01-54-21	5 minutes ago		demo-trials: demo1, demo-experiments: demo1	base-S	5 minutes ago

3. Double-click a trial to display a list of the components in the trial.

Name	Created	Trial component type	Tags	Last modified
TrainingJob	12 minutes ago	Training job		9 minutes ago

4. Double-click one of the components to open the **Describe Trial Component** tab.

Trial stages	Charts	Metrics	Parameters	Artifacts	Settings	Debugger	Trial Mappings												
TrainingJob Created 16 minutes ago			<table border="1"> <thead> <tr> <th>Name</th> <th>MetricID</th> <th>MinValue</th> <th>FinalValue</th> </tr> </thead> <tbody> <tr> <td>validationloss</td> <td>0.06006</td> <td>0.0691</td> <td>0.06797</td> </tr> <tr> <td>trainerror</td> <td>0.033</td> <td>0.07715</td> <td>0.033</td> </tr> </tbody> </table>	Name	MetricID	MinValue	FinalValue	validationloss	0.06006	0.0691	0.06797	trainerror	0.033	0.07715	0.033				
Name	MetricID	MinValue	FinalValue																
validationloss	0.06006	0.0691	0.06797																
trainerror	0.033	0.07715	0.033																

5. On the **Describe Trial Component** tab, choose any of the following column headings to see available information about each trial component:

- **Charts** – Build your own charts.
- **Metrics** – Metrics that are logged by a `Tracker` during a trial run.
- **Parameters** – Hyperparameter values and instance information.
- **Artifacts** – Amazon S3 bucket storage locations for the input dataset and the output model.
- **Amazon Settings** – Job name, ARN, status, creation time, training time, billable time, instance information, and others.
- **Debugger** – A list of debugger rules and any issues found.
- **Trial Mappings**

For information about comparing Experiments entities, see [View and Compare Amazon SageMaker Experiments, Trials, and Trial Components \(p. 1558\)](#).

## Compare Experiments, Trials, and Trial Components

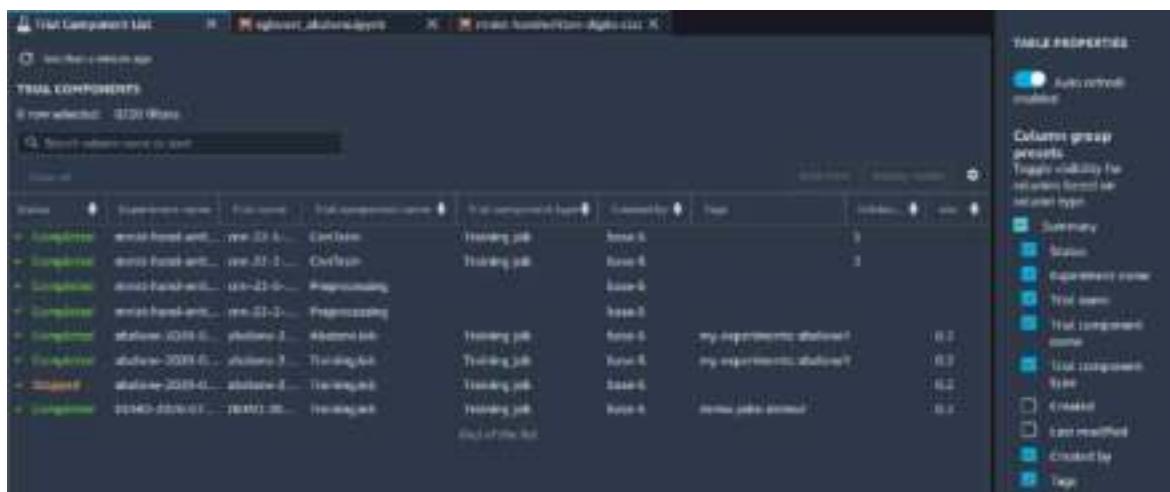
You can compare experiments, trials, and trial components by selecting the entities and opening them in the trial components list. The trial components list is referred to as the Studio Leaderboard. In the Leaderboard you can do the following:

- View detailed information about the entities
- Compare entities

- Stop a training job
- Deploy a model

## To compare experiments, trials, and trial components

1. In the left sidebar of SageMaker Studio, choose the **SageMaker Experiment List** icon (  ).
2. In the **Experiments** browser, choose either the experiment or trial list. For more information, see [View Experiments, Trials, and Trial Components \(p. 1559\)](#).
3. Choose the experiments or trials that you want to compare, right-click the selection, and then choose **Open in trial component list**. The Leaderboard opens and lists the associated Experiments entities as shown in the following screenshot.



The Leaderboard has a **TABLE PROPERTIES** pane on the right side. Use the **Settings** icon (  ) to open and close the pane. You can hide or display properties by category or by individual columns. When you display a chart, the pane changes to display chart properties.

For information on searching the Experiments entities, see [Search Experiments Using Amazon SageMaker Studio \(p. 1567\)](#).

## Track and Compare Tutorial

This tutorial demonstrates how to visually track and compare trials in a model training experiment using Amazon SageMaker Studio. The basis of the tutorial is the [MNIST Handwritten Digits Classification Experiment](#) notebook.

It is intended that this topic be viewed alongside Studio with the MNIST notebook open. As you run through the cells, the sections in this document highlight the relevant code and show you how to observe the results in Studio. Some of the code snippets have been edited for brevity.

To clean up the resources created by the notebook, see [Clean Up Amazon SageMaker Experiment Resources \(p. 1572\)](#).

For a tutorial that showcases additional features of Studio, see [Amazon SageMaker Studio Tour \(p. 51\)](#).

### Prerequisites

- A local copy of the [MNIST](#) example notebook and the companion [mnist.py](#) file. Both files are available from the `sagemaker_experiments/mnist-handwritten-digits-classification-experiment` folder in the [aws/amazon-sagemaker-examples](#) repository. To download the files, choose each link, right-click on the **Raw** button, and then choose **Save as**.
- An Amazon Web Services SSO or IAM account to sign-on to SageMaker Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).

### Topics

- [Open the Notebook in Studio \(p. 1562\)](#)
- [Install the Experiments SDK and Import Modules \(p. 1562\)](#)
- [Transform and Track the Input Data \(p. 1563\)](#)
- [Create and Track an Experiment \(p. 1563\)](#)
- [Compare and Analyze Trials \(p. 1565\)](#)

## Open the Notebook in Studio

### To open the notebook

1. Sign-on to Studio.
2. In the left sidebar, choose the **File Browser** icon (  ).
3. At the top of the file browser pane, choose the **Up arrow** icon and then a **File Upload** dialog opens. Browse to and choose your local versions of the `mnist-handwritten-digits-classification-experiment.ipynb` and `mnist.py` files, and then choose **Open**.
4. The two files are listed in the file browser. Double-click the uploaded notebook file to open the notebook in a new tab.
5. At the top right of the notebook, make sure the kernel is **Python 3 (Data Science)**. If not, choose the current kernel name to open the **Select Kernel** dropdown. Choose **Python 3 (Data Science)** and then choose **Select**.

## Install the Experiments SDK and Import Modules

The [Amazon SageMaker Experiments Python SDK](#) is separate from the [Amazon SageMaker Python SDK](#), which comes preinstalled in SageMaker Studio. Run the first few cells in the notebook to install the Experiments SDK and import the Experiments modules. The relevant sections of the notebook cells are displayed below.

```
import sys
!{sys.executable} -m pip install sagemaker-experiments
```

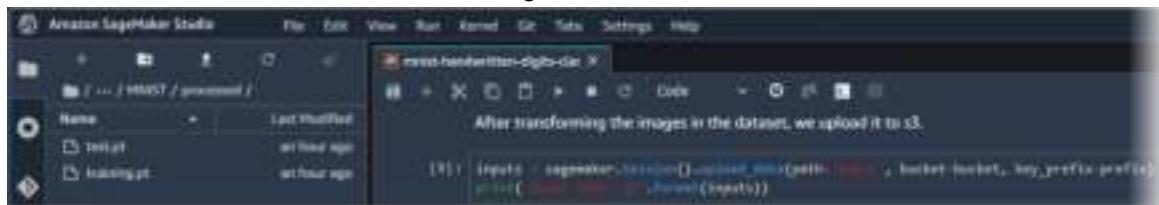
```
import sagemaker
from sagemaker import get_execution_role
from sagemaker.session import Session
from sagemaker.analytics import ExperimentAnalytics

from smexperiments.experiment import Experiment
from smexperiments.trial import Trial
from smexperiments.trial_component import TrialComponent
from smexperiments.tracker import Tracker
```

## Transform and Track the Input Data

The next few cells create an Amazon S3 bucket and a folder in the bucket named *mnist*. In Studio, the file browser displays the *mnist* folder. The input data is downloaded to the *mnist/MNIST/raw* folder, normalized, and then the transformed data is uploaded to the *mnist/MNIST/processed* folder. You can drill down into the *mnist* folder to display, but not open, the data files.

Your screen should look similar to the following:



The last cell in the **Dataset** section creates a **Tracker** for the transform job. The tracker logs the normalization parameters and the URI of the Amazon S3 bucket where the transformed dataset is stored. In a later section, we show how to find this information in Studio. In the next section, the tracker is used to track the experiment and trial runs.

```
with Tracker.create(display_name="Preprocessing", sagemaker_boto_client=sm) as tracker:  
    tracker.log_parameters(  
        {"normalization_mean": 0.1307,  
         "normalization_std": 0.3081,  
     })  
    tracker.log_input(name="mnist-dataset", media_type="s3/uri", value=inputs)
```

## Create and Track an Experiment

The following procedure creates and tracks an experiment to determine the effect of the model's `num_hidden_channel` hyperparameter. As part of the experiment, five trials are created inside a loop, one for each value of the `num_hidden_channel` hyperparameter. Later in the notebook, you'll compare the results of these five trials.

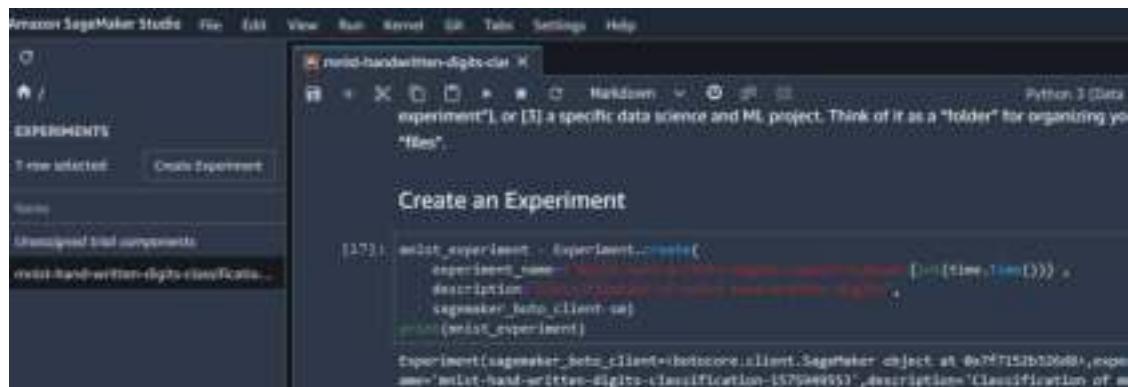
1. In the left sidebar of Studio, choose the **SageMaker Components and registries** icon (💡). In the dropdown menu, choose **Experiments and trials** to display the list of experiments in your account.
2. Run the following cell.

```
mnist_experiment = Experiment.create(  
    experiment_name=f"mnist-hand-written-digits-classification-{int(time.time())}",  
    description="Classification of mnist hand-written digits",  
    sagemaker_boto_client=sm)  
print(mnist_experiment)
```

Output:

```
Experiment(sagemaker_boto_client=<botocore.client.SageMaker object at 0x7f7152b326d8>,  
          experiment_name='mnist-hand-written-digits-classification-1575947870',  
          description='Classification of mnist hand-written digits',  
          experiment_arn='arn:aws:sagemaker:us-east-2:acct-id:experiment/mnist-hand-  
written-digits-classification-1575947870')
```

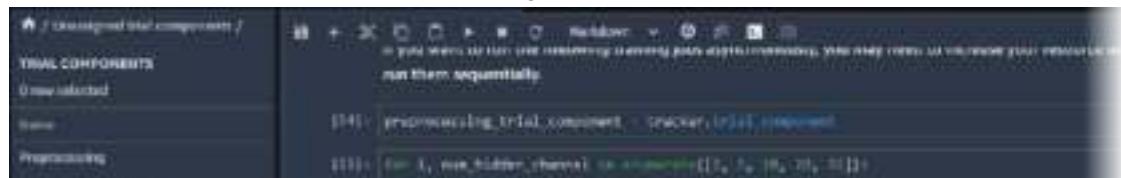
After the code runs, the experiments list contains an entry for the experiment. It might take a moment to display and you might have to refresh the experiments list. Your screen should look similar to the following:



- Run the following cell.

```
preprocessing_trial_component = tracker.trial_component
```

After the code runs, the experiments list contains an entry labeled **Unassigned trial components**. The trial component entry is the data preprocessing step previously created. Double-click the trial component for verification. The trial component isn't associated with an experiment at this time. Your screen should look similar to the following:



- Choose the Home icon in the navigation breadcrumb at the top on the experiments browser. From there, double-click on your experiment to display a list of the trials in the experiment.
- The following code creates trials for the experiment. Each trial trains a model using a different number for the `num_hidden_channel` hyperparameter. The preprocessing trial component is added to each trial for complete tracking (for example, for auditing purposes). The code also specifies definitions for the following metrics:
  - Train loss
  - Test loss
  - Test accuracy

The definitions tell SageMaker to capture those metrics from the algorithm's log output. The metrics are used later to evaluate and compare the models.

```
preprocessing_trial_component = tracker.trial_component

for i, num_hidden_channel in enumerate([2, 5, 10, 20, 32]):
    trial_name = f"cnn-training-job-{num_hidden_channel}-hidden-channels-{int(time.time())}"
    cnn_trial = Trial.create(
        trial_name=trial_name,
        experiment_name=mnist_experiment.experiment_name,
        sagemaker_boto_client=sm,
    )
    hidden_channel_trial_name_map[num_hidden_channel] = trial_name

    cnn_trial.add_trial_component(preprocessing_trial_component)
```

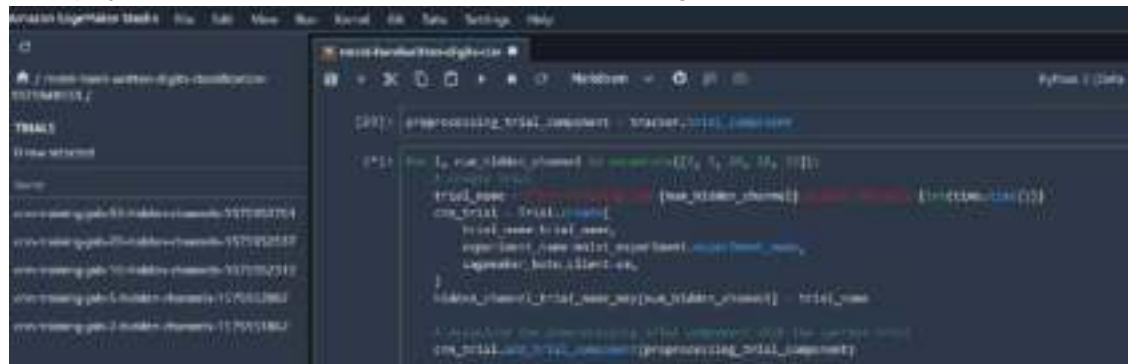
```

estimator = PyTorch(
    py_version='py3',
    framework_version='1.1.0',
    ...,
    hyperparameters={
        'hidden_channels': num_hidden_channel,
        ...
    },
    metric_definitions=[
        {'Name':'train:loss', 'Regex':'Train Loss: (.*)';'},
        {'Name':'test:loss', 'Regex':'Test Average loss: (.*)','},
        {'Name':'test:accuracy', 'Regex':'Test Accuracy: (.*)%'}
    ],
    enable_sagemaker_metrics=True,
)
cnn_training_job_name = "cnn-training-job-{}".format(int(time.time()))

estimator.fit(
    inputs={'training': inputs},
    job_name=cnn_training_job_name,
    experiment_config={
        "TrialName": cnn_trial.trial_name,
        "TrialComponentDisplayName": "Training",
    },
)

```

The trial list automatically updates as each training job runs. It takes a few minutes for each trial to be displayed. Your screen should look similar to the following:



## Compare and Analyze Trials

This section deviates from the notebook and shows you how to compare and analyze the trained models using the SageMaker Studio UI.

### To view the details of a trial

1. Double-click one of the trials to display a list of the trial components associated with the trial. There's a preprocessing job and training job for each trial. Double-click one of the components to open a new tab that displays information about each component.
2. Under **Trial stages**, choose **Preprocessing**. On the **Describe Trial Component** menu, choose **Parameters** to display the normalization parameters that were previously logged. Next, choose **Artifacts** to display the URI of the Amazon S3 bucket where the transformed dataset was stored.
3. Under **Trial stages**, choose **Training**. On the **Describe Trial Component** menu, choose the following items to display information about the training job trial component.

- **Metrics** – test:loss, test:accuracy, and train:loss
- **Parameters** – hyperparameter values and instance information
- **Artifacts** – Amazon S3 storage for the input dataset and the output model
- **Amazon Settings** – job name, ARN, status, creation time, training time, billable time, instance information, and others

### To view a list of trials ordered by test:accuracy

1. Choose the experiment name on the navigation breadcrumb above **TRIAL COMPONENTS** to display the trial list.
2. Choose all five trials. Hold down the CTRL/CMD key and select each trial. Right-click the selection and then choose **Open in trial component list**. A new tab opens that displays each trial and trial component.
3. If the **TABLE PROPERTIES** pane isn't open, choose the **Settings** icon (⚙) in the upper right corner to open it. Deselect everything except **Trial**, **Metrics**, and **Training job**. Choose the **Settings** icon to close the pane.
4. Choose the **test:accuracy** column header to sort the list by decreasing maximum test accuracy. Your screen should look similar to the following:

Trial	Status	test:accuracy	train:loss
cnn-training-job-10-hidden-c...	Completed	0.97	0.21908
cnn-training-job-20-hidden-c...	Completed	0.97	0.2272
cnn-training-job-3-hidden-ch...	Completed	0.97	0.157259
cnn-training-job-52-hidden-c...	Completed	0.97	0.101464
cnn-training-job-5-hidden-ch...	Completed	0.96	0.157844

**TABLE PROPERTIES**

Table properties are global and column level filters will override them. To access filters, click the ⚙ icon.

Auto refresh enabled

**Column group presets**  
Toggle visibility for columns based on column type.

Summary columns:

- Status
- Experiment
- Type
- Trial
- Trial component
- Created on
- Last modified

Metrics

Parameters

Tags

**Type filter**  
Toggle trial component type to apply filter.

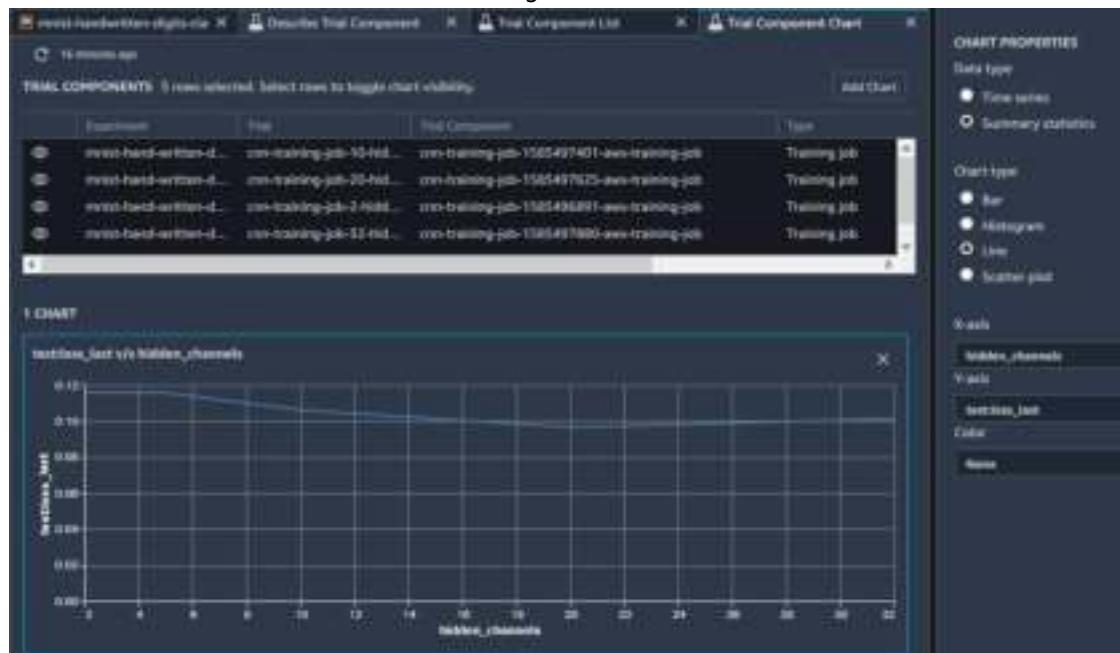
Training job

### To view a chart of test:loss versus num\_hidden\_channel

1. In the **TRIAL COMPONENTS** pane, choose all five trials and then choose **Add chart**. Select inside the chart area to open the preferences pane for **CHART PROPERTIES**.
2. In **CHART PROPERTIES**, choose the following:
  - **Data type - Summary statistics**
  - **Chart type - Line**
  - **X-axis - hidden-channels**
  - **Y-axis - test:loss\_last**

- Color - None

Your screen should look similar to the following:



## Search Experiments Using Amazon SageMaker Studio

You can search your experiments, trials, and trial components using the experiments browser. After choosing the entities, you can search on properties of these entities on the Amazon SageMaker Leaderboard.

### Topics

- [Search Experiments, Trials, and Trial Components \(p. 1567\)](#)
- [Search the SageMaker Studio Leaderboard \(p. 1568\)](#)
- [Search by Tag \(p. 1571\)](#)

## Search Experiments, Trials, and Trial Components

You can search and create multiple filters to display a subset of experiments, trials, and trial components in the experiments browser. After you search and filter your list, you can open the entities in the Leaderboard and search on additional properties.

### To search for experiments, trials, and trial components, and apply filters

For more information about the following steps, along with screenshots, see [Search the SageMaker Studio Leaderboard \(p. 1568\)](#).

1. In the experiments browser, navigate to the entity type you want to search for: experiment, trial, or trial component.
2. Place your cursor in the search box to display the list of column that are searchable.
3. Choose the column that you want to search on and one of the following dialog boxes opens that's specific to the column you chose:

- **String** – Enter a text string in the search dialog box that opens. Enter a string of at least three characters that's part of the component's property value. This filter limits the list to those components with a property value that contains the text string.
  - **Discrete values** – Choose one or more property values from the list. This filter limits the list to those components with a property value that matches at least one of the chosen values.
  - **Date** – Enter a date in **From Date** and **To Date**, or select the dates from the calendar. This filter limits the list to those components that were created or last modified in the specified data range.
  - **Tag** – In **Search tag key**, start entering the tag's key. A list of tag keys is filtered to only those keys with a name that starts with the text string that you enter. Choose the tag key that you want to search on from the list. In **Search tag value**, enter the complete tag value, and then choose **Apply**.
4. To create the filter, choose **Apply**.
  5. To apply additional filters, repeat the preceding steps. You can have a maximum of 20 filters. An entity must match all filters to be displayed.

## Search the SageMaker Studio Leaderboard

The Amazon SageMaker Studio trial components list is referred to as the Leaderboard. You can use the Leaderboard to compare your trials and experiments. The Leaderboard lists properties of the trials, such as the trial status, debugger status, tags, metrics, hyperparameters, and input and output artifacts.

You can search the Leaderboard to find specific trial components. You apply the results of a search to create a filter that displays only those components that satisfy the filter. You can apply up to 20 filters. A component is displayed only if it matches all filters. To remove a filter, choose the X that displays to the right of the filter.

### To search the Leaderboard and apply filters

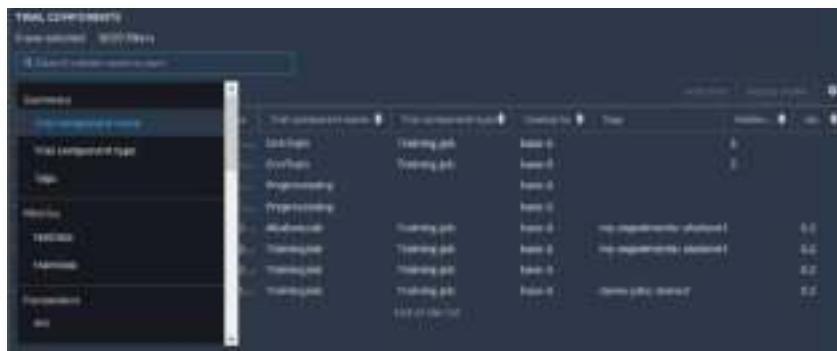
1. If the **TABLE PROPERTIES** pane isn't open in SageMaker Studio, choose the **Settings** icon ( ) in the upper-right corner to open it.
2. In the **Settings** pane, select the check boxes for the columns that you want to view, and then choose the **Settings** icon to close the pane.
3. Place your cursor in the **Search column name** box to display the following list of **TABLE PROPERTIES** column names that are searchable:

#### Summary section

- Trial component name
- Trial component type
- Created
- Last modified
- Created by
- Tags

#### Detail sections (all columns)

- Metrics
- Parameters
- Input artifacts
- Output artifacts



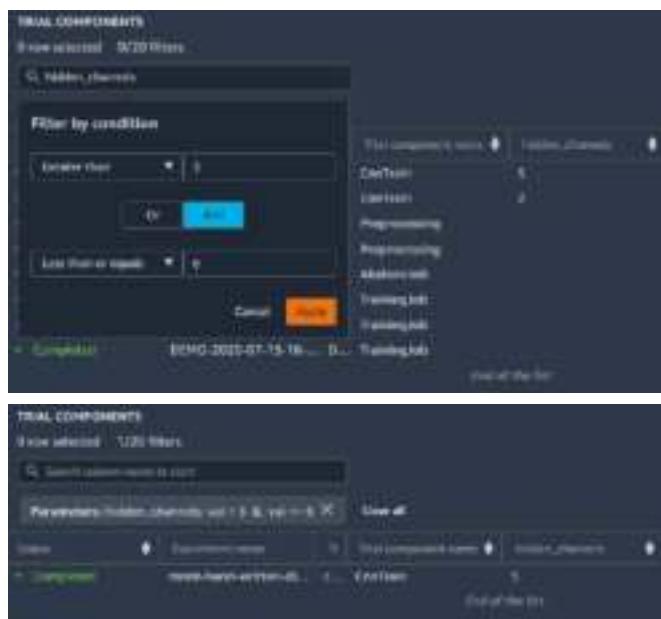
4. To limit the column names when you search the detail sections, start typing the column name. The list displays only those columns with a name that starts with the text string that you enter.
5. Choose the column that you want to search on. This opens a dialog that's specific to the type of data in that column.
6. Enter your search criteria in the dialog for the appropriate data type:
  - **String** – Enter a text string in the search dialog box that opens. Enter a string of at least three characters that's part of the component's property value. This filter limits the list to those components with a property value that contains the text string.

- **Discrete values** – Choose one or more of the property values from the list. This filter limits the list to those components with a property value that matches at least one of the chosen values. For the **Trial component type** column, **Others** matches components with no value in the column.

The screenshot shows the 'TRIAL COMPONENTS' search interface. At the top, there's a search bar with placeholder text 'Search components or tags'. Below it is a 'Filter by values' section with a dropdown menu set to 'Created by' and two options: 'Me' (selected) and 'Everyone else'. A 'Next' button is located at the bottom right of this section. To the right of the filter is a table titled 'Trial components table' with columns 'Trial component name', 'Type', and 'Last updated'. The table lists several entries, all of which are 'Training job' type and were last updated on '2020-07-01'. The first entry is 'abalone-2020-07-01...'. The table has a 'Edit filters' link at the bottom right.

- Date** – Enter a date in **From Date** and **To Date** or select the dates from the calendar. This filter limits the list to those components that were created or last modified in the specified data range.
- Created by** – Select **Me** or **Everyone else**. This filter limits the list to those components that were created by the currently signed-in user or everyone but the currently signed-in user.
- Tag** – Choose the tag to search. For more information, see [Search by Tag \(p. 1571\)](#).
- Detail columns** – Specify the conditions that the component's property value must satisfy.

This screenshot shows the same 'TRIAL COMPONENTS' search interface as above, but with a different filter applied. In the 'Filter by values' dropdown, 'Detail columns' is selected. The table below shows the same list of training jobs, but now includes an additional column 'Detail condition' which contains the value '1' for all rows. The table has a 'Edit filters' link at the bottom right.



7. To create the filter, choose **Apply**.
8. To apply additional filters, repeat the preceding steps. A component must match all filters to be displayed. You can have a maximum of 20 filters.

## Search by Tag

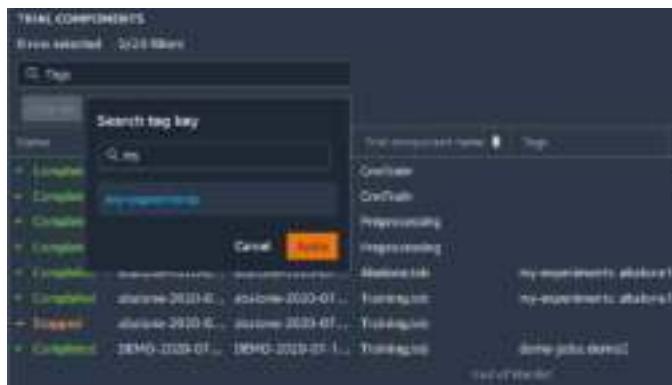
You can add searchable tags to experiments, trials, and trial components when the entities are created or afterwards using the [AddTags](#) API. A tag consists of a unique case-sensitive key and an optional value. Multiple tags can be added to an entity. You can add the same tag to multiple entities.

### To search by tag and apply filters

1. If the **TABLE PROPERTIES** pane isn't open in SageMaker Studio, choose the **Settings** icon (⚙) in the upper-right corner to open it.
2. In the **Settings** pane, select **Tags** if it isn't selected, and then choose the **Settings** icon to close the pane.
3. Place your cursor in the **Search column name** box and then choose **Tags**.



4. In **Search tag key**, start entering the tag's key. A list of tag keys displays only those keys with a name that starts with the text string that you enter.



5. Choose the tag key that you want to search on from the list.
6. In **Search tag value**, enter the complete tag value, and then choose **Apply**.



7. The trial component list displays only those components that have tags that match the key-value pair you chose.



8. To add more tag filters, repeat the previous steps. A tag must match all filters for the component to be displayed.

## Clean Up Amazon SageMaker Experiment Resources

To avoid incurring unnecessary charges, delete the Amazon SageMaker Experiment resources you no longer need. You can't delete Experiment resources through the SageMaker Management Console or the Amazon SageMaker Studio UI. This topic shows you how to clean up these resources using Boto3 and the Experiments SDK. For more information about the Experiments SDK, see [sagemaker-experiments](#).

To delete the experiment, you must delete all trials in the experiment. To delete a trial, you must remove all trial components from the trial. To delete a trial component, you must remove the component from all trials.

**Note**

Trial components can exist independent of trials and experiments. You do not have to delete them. If you want to reuse them, comment out `tc.delete()` in the code.

**Topics**

- [Clean Up Using the Experiments SDK \(p. 1573\)](#)
- [Clean Up Using the Python SDK \(Boto3\) \(p. 1573\)](#)

## Clean Up Using the Experiments SDK

### To clean up using the Experiments SDK

```
import sys
!{sys.executable} -m pip install sagemaker-experiments
```

```
import time

from smexperiments.experiment import Experiment
from smexperiments.trial import Trial
from smexperiments.trial_component import TrialComponent
```

### Define cleanup\_sme\_sdk

```
def cleanup_sme_sdk(experiment):
    for trial_summary in experiment.list_trials():
        trial = Trial.load(trial_name=trial_summary.trial_name)
        for trial_component_summary in trial.list_trial_components():
            tc = TrialComponent.load(
                trial_component_name=trial_component_summary.trial_component_name)
            trial.remove_trial_component(tc)
            try:
                # comment out to keep trial components
                tc.delete()
            except:
                # tc is associated with another trial
                continue
            # to prevent throttling
            time.sleep(.5)
        trial.delete()
        experiment_name = experiment.experiment_name
        experiment.delete()
        print(f"\nExperiment {experiment_name} deleted")
```

### Call cleanup\_sme\_sdk

```
experiment_to_cleanup = Experiment.load(
    # Use experiment name not display name
    experiment_name="experiment-name")

cleanup_sme_sdk(experiment_to_cleanup)
```

## Clean Up Using the Python SDK (Boto3)

### To clean up using Boto 3

```
import boto3
sm = boto3.Session().client('sagemaker')
```

### Define cleanup\_boto3

```
def cleanup_boto3(experiment_name):
    trials = sm.list_trials(ExperimentName=experiment_name)[ 'TrialSummaries' ]
    print('TrialNames:')
    for trial in trials:
        trial_name = trial['TrialName']
        print(f"\n{trial_name}")

        components_in_trial = sm.list_trial_components(TrialName=trial_name)
        print('\tTrialComponentNames:')
        for component in components_in_trial[ 'TrialComponentSummaries' ]:
            component_name = component['TrialComponentName']
            print(f"\t{component_name}")
            sm.disassociate_trial_component(TrialComponentName=component_name,
                TrialName=trial_name)
        try:
            # comment out to keep trial components
            sm.delete_trial_component(TrialComponentName=component_name)
        except:
            # component is associated with another trial
            continue
        # to prevent throttling
        time.sleep(.5)
    sm.delete_trial(TrialName=trial_name)
    sm.delete_experiment(ExperimentName=experiment_name)
    print(f"\nExperiment {experiment_name} deleted")
```

### Call cleanup\_boto3

```
# Use experiment name not display name
experiment_name = "experiment-name"
cleanup_boto3(experiment_name)
```

## Search Using the Amazon SageMaker Console and API

Developing a machine learning model typically requires extensive experimenting with different datasets, algorithms, and hyperparameter values. To manage up to thousands of machine learning model experiments, use the search capabilities in SageMaker.

You can use SageMaker search to:

- Organize, find, and evaluate training jobs using properties, hyperparameters, performance metrics, or any metadata.
- Find the best performing model by reviewing training job and model metrics, such as training loss or validation accuracy.
- Trace a model's lineage to the training job and its related resources, such as the training datasets.

This topic covers searching from the SageMaker console and the SageMaker API. For information on searching in Amazon SageMaker Studio, see [Search Experiments Using Studio \(p. 1567\)](#).

### Topics

- [Sample Notebooks for Managing ML Experiments \(p. 1575\)](#)
- [Organize, Find, and Evaluate Training Jobs \(Console\) \(p. 1575\)](#)
- [Find and Evaluate Training Jobs \(API\) \(p. 1577\)](#)
- [Verify the Datasets Used by Your Training Jobs \(p. 1578\)](#)

- [Trace Model Lineage \(p. 1578\)](#)

## Sample Notebooks for Managing ML Experiments

For a sample notebook that uses Amazon SageMaker model tracking capability to manage ML experiments, see [Managing ML Experimentation using Amazon SageMaker Model Tracking Capability](#).

For instructions on how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). After you have created a notebook instance and opened it, choose the **SageMaker Examples** tab to see a list of all of the SageMaker samples. The notebook for managing ML experiments is located in the **Advanced Functionality** section. To open a notebook, choose its **Use** tab, and choose **Create copy**. If you have questions, post them on the [Amazon Machine Learning Developer Forum](#).

## Organize, Find, and Evaluate Training Jobs (Console)

To organize training jobs, assign one or more tags to them.

To find a specific training job, model, or resource, use model tracking to search on keywords assigned to any searchable items. *Searchable items* include training jobs, models, hyperparameters, metadata, tags, and URLs. To refine your tracking results, you can search using multiple criteria.

To choose the best model for deployment, evaluate how all models performed against one or more metrics. You can use model tracking results to list, sort, and evaluate the performance of the models in your experiments.

### Topics

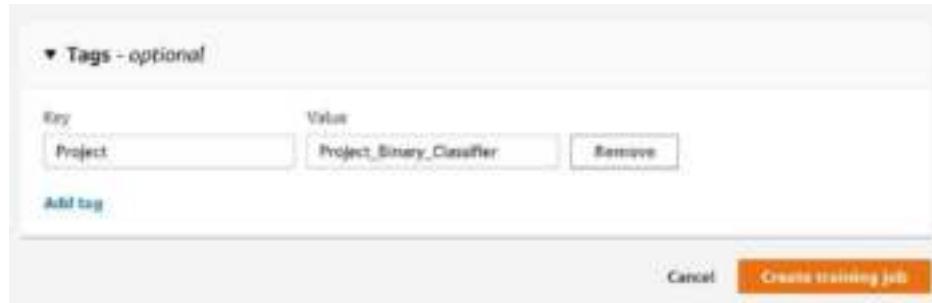
- [Use Tags to Track Training Jobs \(Console\) \(p. 1575\)](#)
- [Find Training Jobs \(Console\) \(p. 1576\)](#)
- [Evaluate Models \(Console\) \(p. 1576\)](#)

## Use Tags to Track Training Jobs (Console)

To group training jobs, create tags with descriptive keys and a value. For example, create tag keys for: project, owner, customer, and industry.

### Add tags to training jobs (console)

1. Open the [Amazon SageMaker console](#).
2. In the navigation pane, choose **Training jobs** and **Create training job**.
3. Scroll to the bottom of the page and enter a key and value for the tag.



4. To add another tag, choose **Add tag**, and add another key-value pair.

## Find Training Jobs (Console)

You can search for training jobs using a variety of job attributes. Note that some search parameters appear only if you have created a training job with that attribute. For example, **Tags** appears only if you have added a tag for a training job.

### To find training jobs (console)

1. Open the [Amazon SageMaker console](#).
2. In the navigation pane, choose **Search**.
3. Add **Parameters**.
  - a. In the search box, enter a parameter and choose a parameter type, for example **TrainingJobName**.
  - b. Choose a conditional operation. For numeric values, use operators such as **is equals to**, **lesser than**, or **or greater than**. For text-based values, use operators such as **equals to** or **contains**.
  - c. Enter a value for the parameter.
4. (Optional) To refine your search, add additional search criteria. Choose **Add row** and enter the parameter values.
5. Choose **Search**.

## Evaluate Models (Console)

To evaluate a model's performance, review its metadata, hyperparameters, and metrics. To highlight metrics, adjust the view to show only metrics and important hyperparameters.

### To evaluate a model (console)

1. Open the [Amazon SageMaker console](#).
2. In the navigation pane, choose **Search** and search for training jobs by specifying relevant parameters. The results are displayed in a table.

Results: Training jobs

HyperParameter mini_batch_size ▾	HyperParameter predictor_type ▾	Metric train:binary_f_beta ▾	Metric train:progress ▾	Metric train:objective_loss
300	binary_classifier	0.966639518737793	100	0.02381423674521
100	binary_classifier	0.9652714133262634	100	0.02350491285324
200	binary_classifier	0.9647442698478699	100	0.02325980737801

3. Open the preferences window by choosing the settings icon in the search results table.

4. To show or hide a hyperparameter or metric, turn it on or off by choosing **Hyperparameter or Metric**.
5. Make necessary changes, then choose **Update view**.
6. After viewing metrics and important hyperparameters, you can compare and contrast the result. Then, you can choose the best model to host or investigate the models that are performing poorly.

## Find and Evaluate Training Jobs (API)

To find and evaluate training jobs or to get suggestions for items used in experiments that are searchable, you can use the [Search API](#).

### Topics

- [Find Training Jobs \(API\) \(p. 1577\)](#)
- [Evaluate Models \(API\) \(p. 1577\)](#)
- [Get Suggestions for a Search \(API\) \(p. 1578\)](#)

## Find Training Jobs (API)

To find training jobs, create a search parameter using the `search_params` parameter. Then use the `search` function in the `smclient` subprocess in the Amazon SDK for Python (Boto3).

The following example shows how to use the [Search API](#) to find training jobs.

```
import boto3

search_params={
    "MaxResults": 10,
    "Resource": "TrainingJob",
    "SearchExpression": {
        "Filters": [
            {
                "Name": "Tags.Project",
                "Operator": "Equals",
                "Value": "Project_Binary_Classifier"
            }]
    },
    "SortBy": "Metrics.train:binary_classification_accuracy",
    "SortOrder": "Descending"
}

smclient = boto3.client(service_name='sagemaker')
results = smclient.search(**search_params)
```

## Evaluate Models (API)

To evaluate models, run a search as described in [Find Training Jobs \(API\) \(p. 1577\)](#), review model metrics, then, use the Amazon SDK for Python (Boto3) to create a table and plot it.

The following example shows how to evaluate models and to display the results in a table.

```
import pandas

headers=[ "Training Job Name", "Training Job Status", "Batch Size", "Binary Classification Accuracy"]
rows=[]
for result in results['Results']:
    trainingJob = result['TrainingJob']
    metrics = trainingJob['FinalMetricDataList']
```

```
rows.append([trainingJob['TrainingJobName'],
    trainingJob['TrainingJobStatus'],
    trainingJob['HyperParameters']['mini_batch_size'],
    metrics[[x['MetricName'] for x in
    metrics].index('train:binary_classification_accuracy')]['Value']
])

df = pandas.DataFrame(data=rows,columns=headers)

from IPython.display import display, HTMLdisplay(HTML(df.to_html()))
```

## Get Suggestions for a Search (API)

To get suggestions for a search, use the [GetSearchSuggestions API](#).

The following example for Amazon SDK for Python (Boto3) is a `get_search_suggestions` request for items containing `linear`.

```
search_suggestion_params={
    "Resource": "TrainingJob",
    "SuggestionQuery": {
        "PropertyNameQuery": {
            "PropertyNameHint": "linear"
        }
    }
}
```

The following is an example response for a `get_search_suggestions` request.

```
{
    'PropertyNameSuggestions': [{PropertyName': 'hyperparameters.linear_init_method'},
        {'PropertyName': 'hyperparameters.linear_init_value'},
        {'PropertyName': 'hyperparameters.linear_init_sigma'},
        {'PropertyName': 'hyperparameters.linear_lr'},
        {'PropertyName': 'hyperparameters.linear_wd'}]
}
```

After getting search suggestions, you can use one of the property names in a search.

## Verify the Datasets Used by Your Training Jobs

You can use model tracking capability to verify which datasets were used in training, where holdout datasets were used, and other details about training jobs. For example, use model tracking capability to verify that a specific dataset was used in a training job for an audit or to verify compliance.

To check whether a specific dataset was used in a training job, you search for the URL to its location in Amazon Simple Storage Service (Amazon S3). Model tracking capability returns the training jobs that used the dataset that you specify. If your search doesn't return the dataset (the result is empty), the dataset wasn't used in a training job. An empty result confirms, for example, that a holdout dataset wasn't used.

## Trace Model Lineage

You can use model tracking capability to get information about the lineage of training jobs and the model resources that were used for them, including the dataset, algorithm, hyperparameters, and metrics. For example, if you find that the performance of a hosted model has declined, you can review its training job and the resources it used to determine what's causing the problem.

## Topics

- [Trace Model Lineage \(Console\) \(p. 1579\)](#)
- [Trace Model Lineage \(API\) \(p. 1579\)](#)

## Trace Model Lineage (Console)

### To trace a model's lineage (console)

1. Open the [Amazon SageMaker console](#).
2. In the navigation pane, choose **Endpoints**, and choose the relevant endpoint.
3. Scroll to the **Endpoint configuration settings** section. This section lists all of the model versions deployed at the endpoint, with a hyperlink to the training job that created each.

## Trace Model Lineage (API)

To trace a model's lineage, get the model's name, then use it to search for training jobs.

The following example shows how to trace a model's lineage using the API.

```
# Get the name of model deployed at endpoint
endpoint_config = smclient.describe_endpoint_config(EndpointConfigName=endpointName)
model_name = endpoint_config['ProductionVariants'][0]['ModelName']

# Get the model's name
model = smclient.describe_model(ModelName=model_name)

# Search the training job by the location of model artifacts in Amazon S3
search_params={
    "MaxResults": 1,
    "Resource": "TrainingJob",
    "SearchExpression": {
        "Filters": [
            {
                "Name": "ModelArtifacts.S3ModelArtifacts",
                "Operator": "Equals",
                "Value": model['PrimaryContainer']['ModelDataUrl']
            }]
    }
}
results = smclient.search(**search_params)
```

After finding the training job, you can review the resources used to train the model.

# Amazon SageMaker Debugger

Debug, monitor, and profile training jobs in real time, detect non-converging conditions, optimize resource utilization by eliminating bottlenecks, improve training time, and reduce costs of your machine learning models using Amazon SageMaker Debugger.

## Amazon SageMaker Debugger Features

A machine learning (ML) training job can have problems such as system bottlenecks, overfitting, saturated activation functions, and vanishing gradients, which can compromise model performance.

SageMaker Debugger profiles and debugs training jobs to help resolve such problems and improve your ML model's compute resource utilization and performance. Debugger offers tools to send alerts when training anomalies are found, take actions against the problems, and identify the root cause of them by visualizing collected metrics and tensors.

SageMaker Debugger supports Apache MXNet, TensorFlow, PyTorch, and XGBoost. For more information about available frameworks and versions, see [Supported Frameworks and Algorithms \(p. 1581\)](#).



The high-level Debugger workflow is as follows:

1. Configure a SageMaker training job with Debugger.
  - Configure using the SageMaker [Estimator API \(for Python SDK\)](#).
  - Configure using the SageMaker [CreateTrainingJob request \(for Boto3 or CLI\)](#).
  - Configure [custom training containers \(p. 1672\)](#) with Debugger.
2. Start a training job and monitor training issues in real time.
  - [SageMaker Studio Debugger dashboards in Studio Experiments and trials \(p. 1685\)](#).
  - [List of Debugger Built-in Rules \(p. 1626\)](#).
3. Get alerts and take prompt actions against the training issues.
  - Receive texts and emails and stop training jobs when training issues are found using [Debugger Built-in Actions for Rules \(p. 1676\)](#).
  - Set up your own actions using [Amazon CloudWatch Events and Amazon Lambda \(p. 1680\)](#).
4. Receive training reports, suggestions to fix the issues, and insights into your training jobs.
  - [Studio Debugger Insights dashboard for deep learning frameworks](#)
  - [Deep learning framework profiling report](#)
  - [SageMaker XGBoost training report](#)
5. Explore deep analysis of the training issues and bottlenecks.
  - For profiling training jobs, see [Analyze Data Using the SMDebug Client Library \(p. 1724\)](#).
  - For debugging model parameters, see [Visualize Debugger Output Tensors in TensorBoard \(p. 1681\)](#).
6. Fix the issues, considering the suggestions provided by Debugger, and repeat steps 1–5 until you optimize your model and achieve target accuracy.

The SageMaker Debugger developer guide walks you through the following topics.

### Topics

- [Supported Frameworks and Algorithms \(p. 1581\)](#)
- [Amazon SageMaker Debugger Architecture \(p. 1582\)](#)
- [Get Started with Debugger Tutorials \(p. 1584\)](#)

- [Configure Debugger Using Amazon SageMaker Python SDK \(p. 1593\)](#)
- [Configure Debugger Using Amazon SageMaker API \(p. 1615\)](#)
- [List of Debugger Built-in Rules \(p. 1626\)](#)
- [Create Debugger Custom Rules for Training Job Analysis \(p. 1670\)](#)
- [Use Debugger with Custom Training Containers \(p. 1672\)](#)
- [Action on Amazon SageMaker Debugger Rules \(p. 1675\)](#)
- [SageMaker Debugger on Studio \(p. 1685\)](#)
- [SageMaker Debugger Interactive Reports \(p. 1700\)](#)
- [Analyze Data Using the SMDebug Client Library \(p. 1724\)](#)
- [Visualize Amazon SageMaker Debugger Output Tensors in TensorBoard \(p. 1732\)](#)
- [Best Practices for Amazon SageMaker Debugger \(p. 1734\)](#)
- [Amazon SageMaker Debugger Advanced Topics and Reference Documentation \(p. 1737\)](#)

## Supported Frameworks and Algorithms

The following table shows SageMaker machine learning frameworks and algorithms supported by Debugger.

SageMaker frameworks and algorithms	Monitoring system bottlenecks	Profiling deep learning framework operations	Debugging model parameters
TensorFlow	All <a href="#">Amazon Deep learning containers</a>	Amazon TensorFlow deep learning containers 2.3.1 or later	<a href="#">Amazon TensorFlow deep learning containers</a> 1.15.4 or later
PyTorch		Amazon PyTorch deep learning containers 1.6.0 or later	<a href="#">Amazon PyTorch deep learning containers</a> 1.5.0 or later
MXNet		-	<a href="#">Amazon MXNet deep learning containers</a> 1.6.0 or later
XGBoost	1.0-1, 1.2-1	-	1.0-1, 1.2-1
SageMaker generic estimator	SageMaker built-in algorithms using image URIs  <a href="#">Custom training containers (p. 1672)</a> (with the Amazon deep learning container images, public Docker images, or your own Docker images)	-  <a href="#">Custom training containers (p. 1672)</a> (available for TensorFlow, PyTorch, MXNet, and XGBoost with manual hook registration)	<a href="#">Custom training containers (p. 1672)</a> (available for TensorFlow, PyTorch, MXNet, and XGBoost with manual hook registration)

- **Monitoring system bottlenecks** – Monitor the system utilization rate for resources such as CPU, GPU, memories, network, and data I/O metrics. This is a framework and model agnostic feature and available for any training jobs in SageMaker.

- **Profiling deep learning framework operations** – Profile the deep learning operations of the TensorFlow and PyTorch frameworks, such as step durations, data loaders, forward and backward operations, Python profiling metrics, and framework-specific metrics.
- **Debugging model parameters** – Track and debug model parameters, such as weights, gradients, biases, and scalar values of your training job. Available deep learning frameworks are Apache MXNet, TensorFlow, PyTorch, and XGBoost.

If the framework or algorithm that you want to train and debug is not listed in the table, go to the [Amazon Discussion Forum](#) and leave feedback on SageMaker Debugger.

## Use Debugger with Custom Training Containers

Bring your training containers to SageMaker and gain insights into your training jobs using Debugger. Maximize your work efficiency by optimizing your model on Amazon EC2 instances using the monitoring and debugging features.

For more information about how to build your training container with the SMDebug client library, push it to the Amazon Elastic Container Registry (Amazon ECR), and monitor and debug, see [Use Debugger with Custom Training Containers \(p. 1672\)](#).

## Debugger Open-Source GitHub Repositories

Debugger APIs are provided through the SageMaker Python SDK and designed to construct Debugger hook and rule configurations for the SageMaker [CreateTrainingJob](#) and [DescribeTrainingJob](#) API operations. The SMDebug client library provides tools to register *hooks* and access the training data through its *trial* feature, all through its flexible and powerful API operations. It supports the machine learning frameworks TensorFlow, PyTorch, MXNet, and XGBoost on Python 3.6 and later.

For direct resources about the Debugger and SMDebug API operations, see the following links:

- [The Amazon SageMaker Python SDK documentation](#)
- [The Amazon SageMaker Debugger APIs](#)
- [The SMDebug open source client library](#)
- [The SMDebug PyPI](#)

If you use the SDK for Java to conduct SageMaker training jobs and want to configure Debugger APIs, see the following references:

- [Amazon SageMaker Debugger API Operations \(p. 1737\)](#)
- [Configure Debugger Using Amazon SageMaker API \(p. 1615\)](#)

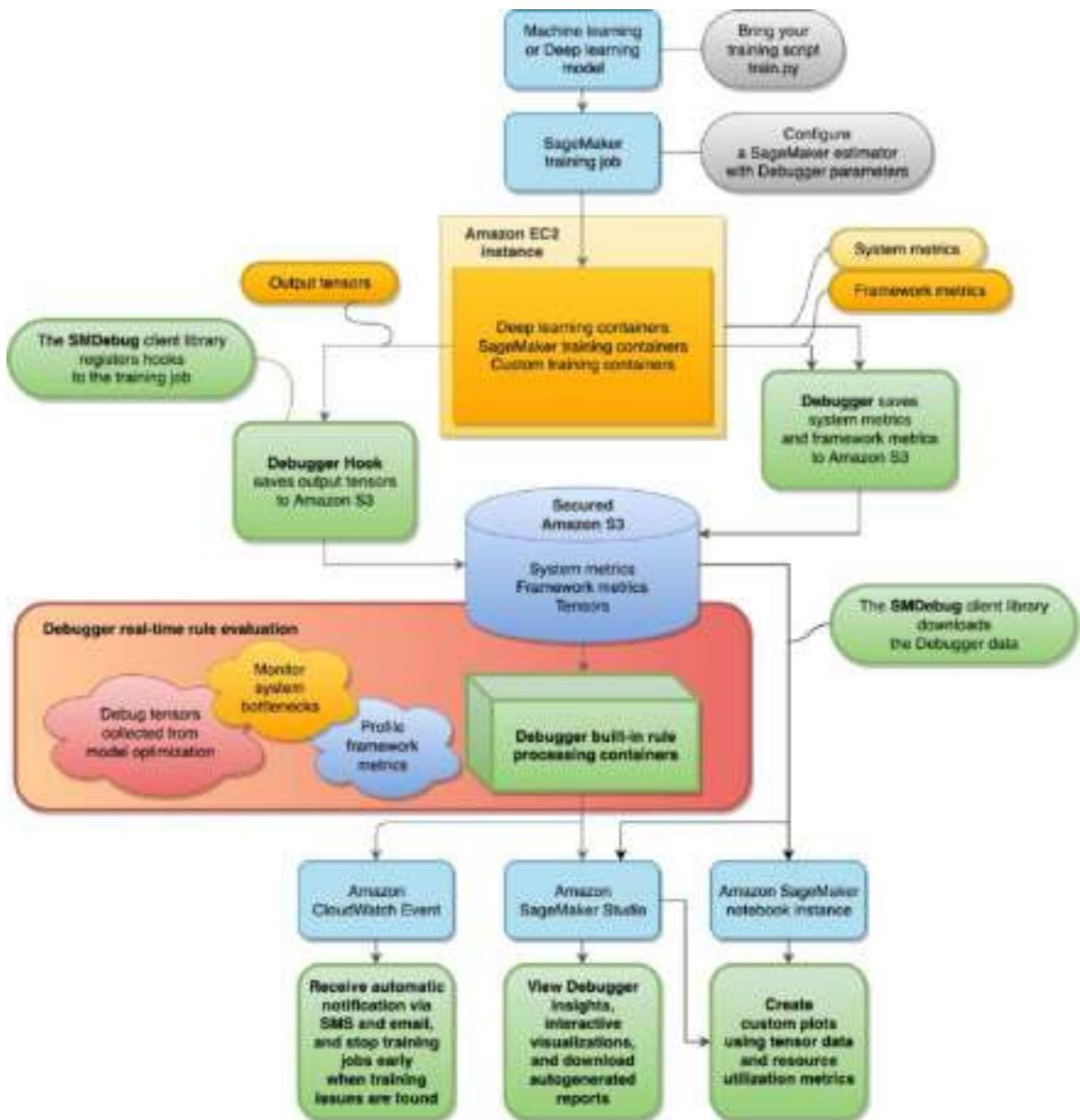
## Amazon SageMaker Debugger Architecture

This topic walks you through a high-level overview of the Amazon SageMaker Debugger workflow.

Debugger supports profiling functionality for *performance optimization* to identify computation issues, such as system bottlenecks and underutilization, and to help optimize hardware resource utilization at scale.

Debugger's debugging functionality for *model optimization* is about analyzing non-converging training issues that can arise while minimizing the loss functions using optimization algorithms, such as gradient descent and its variations.

The following diagram shows the architecture of SageMaker Debugger. The blocks with bold boundary lines are what Debugger manages to analyze your training job.



Debugger stores the following data from your training jobs in your secured Amazon S3 bucket:

- **System metrics** – Hardware resource utilization data, such as CPU, GPU, CPU and GPU memory, network, and data input and output (I/O) metrics.
- **Framework metrics** – Metrics to track each framework operation per call or sampling, such as convolutional layer operations in the forward pass, batch normalization operations in the backward pass, data loader processes between steps, and gradient descent algorithm operations to calculate and update the loss function.
- **Output tensors** – Collections of scalars and model parameters that are continuously updated during the forward and backward passes while training ML models. The output tensors include scalar values (accuracy and loss) and matrices (weights, gradients, input layers, and output layers).

#### Note

By default, Debugger monitors and debugs SageMaker training jobs without any Debugger-specific parameters configured in SageMaker estimators. Debugger collects system metrics

every 500 milliseconds and basic output tensors (scalar outputs such as loss and accuracy) every 500 steps. It also runs the `ProfilerReport` rule to analyze the system metrics and aggregate the Studio Debugger insights dashboard and a profiling report. Debugger saves the output data in your secured Amazon S3 bucket.

The Debugger built-in rules run on processing containers, which are designed to evaluate machine learning models by processing the training data collected in your S3 bucket (see [Process Data and Evaluate Models](#)). The built-in rules are fully managed by Debugger. You can also create your own rules customized to your model to watch for any issues you want to monitor.

See the following topics for best practices to improve the performance of your model using SageMaker Debugger.

## Get Started with Debugger Tutorials

The following topics walk you through tutorials from the basics to advanced use cases of monitoring, profiling, and debugging SageMaker training jobs using Debugger. Explore the Debugger features and learn how you can debug and improve your machine learning models efficiently by using Debugger.

### Topics

- [Debugger Tutorial Videos \(p. 1584\)](#)
- [Debugger Example Notebooks \(p. 1585\)](#)
- [Debugger Advanced Demos and Visualization \(p. 1587\)](#)

## Debugger Tutorial Videos

The following videos provide a tour of Amazon SageMaker Debugger capabilities using SageMaker Studio and SageMaker notebook instances.

### Topics

- [Analyze, Detect, and Get Alerted on Problems with Training Runs Using Amazon SageMaker Debugger \(p. 1584\)](#)
- [Debug Models with Amazon SageMaker Debugger in Studio \(p. 1584\)](#)
- [Deep Dive on Amazon SageMaker Debugger and SageMaker Model Monitor \(p. 1585\)](#)

## Analyze, Detect, and Get Alerted on Problems with Training Runs Using Amazon SageMaker Debugger

*Emily Webber, Amazon Machine Learning Specialist | Length: 13 minutes 54 seconds*

This tutorial video gives you a tour of Amazon SageMaker Debugger to capture, debug, and visualize model output data from a training model with MXNet. Learn how Amazon SageMaker Debugger makes the training process transparent by automatically capturing metrics, analyzing training runs, and detecting problems.

### [Analyze, Detect, and Get Alerted on Problems with Training Runs Using Amazon SageMaker Debugger](#)

You can find the example notebook in this video at [Visualizing Debugging Tensors of MXNet training](#) in the [Amazon SageMaker Examples](#) GitHub repository.

## Debug Models with Amazon SageMaker Debugger in Studio

*Julien Simon, Amazon Technical Evangelist | Length: 14 minutes 17 seconds*

This tutorial video demonstrates how to use Amazon SageMaker Debugger to capture and inspect debugging information from a training model. The example training model used in this video is a simple convolutional neural network (CNN) based on Keras with the TensorFlow backend. SageMaker in a TensorFlow framework and Debugger enable you to build an estimator directly using the training script and debug the training job.

### [Debug Models with Amazon SageMaker Debugger \(part 1\)](#)

You can find the example notebook in the video in [this Studio Demo repository](#) provided by the author. You need to clone the `debugger.ipynb` notebook file and the `mnist_keras_tf.py` training script to your SageMaker Studio or a SageMaker notebook instance. After you clone the two files, specify the path `keras_script_path` to the `mnist_keras_tf.py` file inside the `debugger.ipynb` notebook. For example, if you cloned the two files in the same directory, set it as `keras_script_path = "mnist_keras_tf.py"`.

## [Deep Dive on Amazon SageMaker Debugger and SageMaker Model Monitor](#)

*Julien Simon, Amazon Technical Evangelist | Length: 44 minutes 34 seconds*

This video session explores advanced features of Debugger and SageMaker Model Monitor that help boost productivity and the quality of your models. First, this video shows how to detect and fix training issues, visualize tensors, and improve models with Debugger. Next, at 22:41, the video shows how to monitor models in production and identify prediction issues such as missing features or data drift using SageMaker Model Monitor. Finally, it offers cost optimization tips to help you make the most of your machine learning budget.

### [Debug Models with Debugger \(part 2\)](#)

You can find the example notebook in the video in [this Amazon Dev Days 2020 repository](#) offered by the author.

## Debugger Example Notebooks

SageMaker Debugger example notebooks are provided in the [aws/amazon-sagemaker-examples](#) repository. The Debugger example notebooks walk you through basic to advanced use cases of debugging and profiling training jobs.

We recommend that you run the example notebooks on SageMaker Studio or a SageMaker Notebook instance because most of the examples are designed for training jobs in the SageMaker ecosystem, including Amazon EC2, Amazon S3, and Amazon SageMaker Python SDK.

To clone the example repository to SageMaker Studio, follow the instructions at [Amazon SageMaker Studio Tour](#).

To find the examples in a SageMaker Notebook instance, follow the instructions at [SageMaker Notebook Instance Example Notebooks](#).

### Important

To use the new Debugger features, you need to upgrade the SageMaker Python SDK and the SMDebug client library. In your iPython kernel, Jupyter Notebook, or JupyterLab environment, run the following code to install the latest versions of the libraries and restart the kernel.

```
import sys
import IPython
!{sys.executable} -m pip install -U sagemaker smdebug
IPython.Application.instance().kernel.do_shutdown(True)
```

## [Debugger Example Notebooks for Profiling Training Jobs](#)

The following list shows Debugger example notebooks introducing Debugger's adaptability to monitor and profile training jobs for various machine learning models, datasets, and frameworks.

Notebook Title	Framework	Model	Dataset	Description
<a href="#">Amazon SageMaker Debugger Profiling Data Analysis</a>	TensorFlow	Keras ResNet50	Cifar-10	This notebook provides an introduction to interactive analysis of profiled data captured by SageMaker Debugger. Explore the full functionality of the SMDebug interactive analysis tools.
<a href="#">Profile machine learning training with Amazon SageMaker Debugger</a>	TensorFlow	1-D Convolutional Neural Network	IMDB dataset	Profile a TensorFlow 1-D CNN for sentiment analysis of IMDB data that consists of movie reviews labeled as having positive or negative sentiment. Explore the Studio Debugger insights and Debugger profiling report.
<a href="#">Profiling TensorFlow ResNet model training with various distributed training settings</a>	TensorFlow	ResNet50	Cifar-10	Run TensorFlow training jobs with various distributed training settings, monitor system resource utilization, and profile model performance using Debugger.
<a href="#">Profiling PyTorch ResNet model training with various distributed training settings</a>	PyTorch	ResNet50	Cifar-10	Run PyTorch training jobs with various distributed training settings, monitor system resource utilization, and profile model performance using Debugger.

## Debugger Example Notebooks for Analyzing Model Parameters

The following list shows Debugger example notebooks introducing Debugger's adaptability to debug training jobs for various machine learning models, datasets, and frameworks.

Notebook Title	Framework	Model	Dataset	Description
<a href="#">Amazon SageMaker Debugger - Use built-in rule</a>	TensorFlow	Convolutional Neural Network	MNIST	Use the Amazon SageMaker Debugger built-in rules for debugging a TensorFlow model.
<a href="#">Amazon SageMaker Debugger - Tensorflow 2.1</a>	TensorFlow	ResNet50	Cifar-10	Use the Amazon SageMaker Debugger hook configuration and built-in rules for debugging

Notebook Title	Framework	Model	Dataset	Description
				a model with the Tensorflow 2.1 framework.
<a href="#">Visualizing Debugging Tensors of MXNet training</a>	MXNet	Gluon Convolutional Neural Network	Fashion MNIST	Run a training job and configure SageMaker Debugger to store all tensors from this job, then visualize those tensors in a notebook.
<a href="#">Enable Spot Training with Amazon SageMaker Debugger</a>	MXNet	Gluon Convolutional Neural Network	Fashion MNIST	Learn how Debugger collects tensor data from a training job on a spot instance, and how to use the Debugger built-in rules with managed spot training.
<a href="#">Debugging XGBoost Training Jobs with Amazon SageMaker Debugger Using Rules</a>	XGBoost	XGBoost Regression	Abalone	Learn how to use the Debugger hook and built-in rules for collecting and visualizing tensor data from an XGBoost regression model, such as loss values, features, and SHAP values.

To find advanced visualizations of model parameters and use cases, see the next topic at [Debugger Advanced Demos and Visualization \(p. 1587\)](#).

## Debugger Advanced Demos and Visualization

The following demos walk you through advanced use cases and visualization scripts using Debugger.

### Topics

- [Train and Tune Your Models with Amazon SageMaker Experiments and Debugger \(p. 1587\)](#)
- [Using SageMaker Debugger to Monitor a Convolutional Autoencoder Model Training \(p. 1590\)](#)
- [Using SageMaker Debugger to Monitor Attentions in BERT Model Training \(p. 1590\)](#)
- [Using SageMaker Debugger to Visualize Class Activation Maps in Convolutional Neural Networks \(CNNs\) \(p. 1593\)](#)

## Train and Tune Your Models with Amazon SageMaker Experiments and Debugger

*Dr. Nathalie Rauschmayr, Amazon Applied Scientist | Length: 49 minutes 26 seconds*

### [Train and Prune Models with SageMaker Experiments and Debugger](#)

Find out how Amazon SageMaker Experiments and Debugger can simplify the management of your training jobs. Amazon SageMaker Debugger provides transparent visibility into training jobs and saves training metrics into your Amazon S3 bucket. SageMaker Experiments enables you to call the training information as *trials* through SageMaker Studio and supports visualization of the training job. This helps you keep model quality high while reducing less important parameters based on importance rank.

This video demonstrates a *model pruning* technique that makes pre-trained ResNet50 and AlexNet models lighter and affordable while keeping high standards for model accuracy.

SageMaker Estimator trains those algorithms supplied from the PyTorch model zoo in an Amazon Deep Learning Containers with PyTorch framework, and Debugger extracts training metrics from the training process.

The video also demonstrates how to set up a Debugger custom rule to watch the accuracy of a pruned model, to trigger an Amazon CloudWatch event and an Amazon Lambda function when the accuracy hits a threshold, and to automatically stop the pruning process to avoid redundant iterations.

Learning objectives are as follows:

- Learn how to use SageMaker to accelerate ML model training and improve model quality.
- Understand how to manage training iterations with SageMaker Experiments by automatically capturing input parameters, configurations, and results.
- Discover how Debugger makes the training process transparent by automatically capturing real-time tensor data from metrics such as weights, gradients, and activation outputs of convolutional neural networks.
- Use CloudWatch to trigger Lambda when Debugger catches issues.
- Master the SageMaker training process using SageMaker Experiments and Debugger.

You can find the notebooks and training scripts used in this video from [SageMaker Debugger PyTorch Iterative Model Pruning](#).

The following image shows how the iterative model pruning process reduces the size of AlexNet by cutting out the 100 least significant filters based on importance rank evaluated by activation outputs and gradients.

The pruning process reduced the initial 50 million parameters to 18 million. It also reduced the estimated model size from 201 MB to 73 MB.

**Pruning iteration: 0**

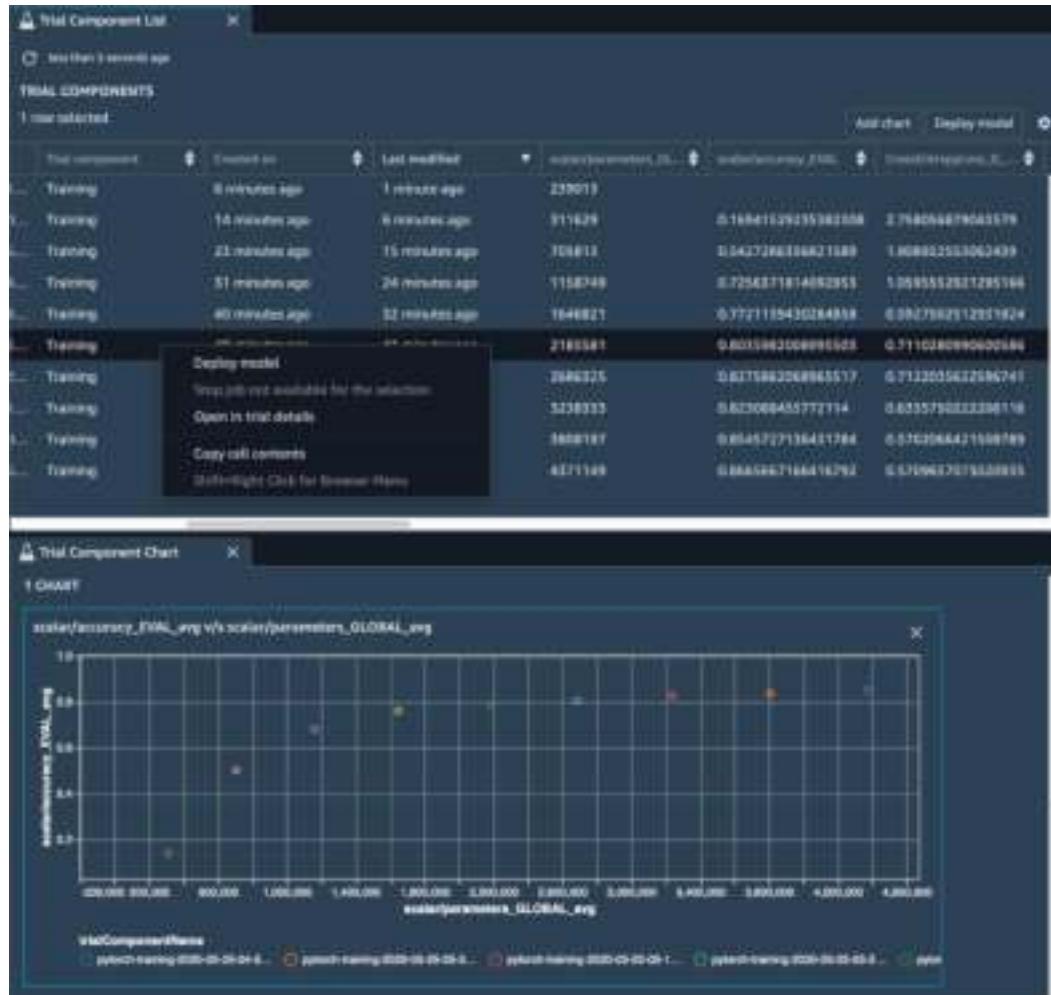
Layer (type)	Output Shape	Param #
Conv2d-1	[1, 58, 55, 55]	21,112
ReLU-2	[1, 58, 55, 55]	0
MaxPool2d-3	[1, 58, 27, 27]	0
Conv2d-4	[1, 166, 27, 27]	240,866
ReLU-5	[1, 166, 27, 27]	0
MaxPool2d-6	[1, 166, 13, 13]	0
Conv2d-7	[1, 305, 13, 13]	455,975
ReLU-8	[1, 305, 13, 13]	0
Conv2d-9	[1, 206, 13, 13]	565,676
ReLU-10	[1, 206, 13, 13]	0
Conv2d-11	[1, 217, 13, 13]	402,535
ReLU-12	[1, 217, 13, 13]	0
MaxPool2d-13	[1, 217, 6, 6]	0
AdaptiveAvgPool2d-14	[1, 217, 6, 6]	0
Dropout-15	[1, 7812]	0
Linear-16	[1, 4096]	32,882,818
ReLU-17	[1, 4096]	0
Dropout-18	[1, 4096]	0
Linear-19	[1, 4096]	16,781,312
ReLU-20	[1, 4096]	0
Linear-21	[1, 101]	413,797
<hr/>		
Total params:	50,883,321	
Trainable params:	50,883,321	
Non-trainable params:	0	

---

Input size (MB):	0.57
Forward/backward pass size (MB):	7.27
Params size (MB):	194.18
Estimated Total Size (MB):	201.95

---

You also need to track model accuracy, and the following image shows how you can plot the model pruning process to visualize changes in model accuracy based on the number of parameters in SageMaker Studio.



In SageMaker Studio, choose the **Experiments** tab, select a list of tensors saved by Debugger from the pruning process, and then compose a **Trial Component List** panel. Select all ten iterations and then choose **Add chart** to create a **Trial Component Chart**. After you decide on a model to deploy, choose the trial component and choose a menu to perform an action or choose **Deploy model**.

#### Note

To deploy a model through SageMaker Studio using the following notebook example, add a line at the end of the `train` function in the `train.py` script.

```
# In the train.py script, look for the train function in line 58.
def train(epochs, batch_size, learning_rate):
    ...
    print('acc:{:.4f}'.format(correct/total))
    hook.save_scalar("accuracy", correct/total, sm_metric=True)

    # Add the following code to line 128 of the train.py script to save the pruned
    # models
    # under the current SageMaker Studio model directory
    torch.save(model.state_dict(), os.environ['SM_MODEL_DIR'] + '/model.pt')
```

## Using SageMaker Debugger to Monitor a Convolutional Autoencoder Model Training

This notebook demonstrates how SageMaker Debugger visualizes tensors from an unsupervised (or self-supervised) learning process on a MNIST image dataset of handwritten numbers.

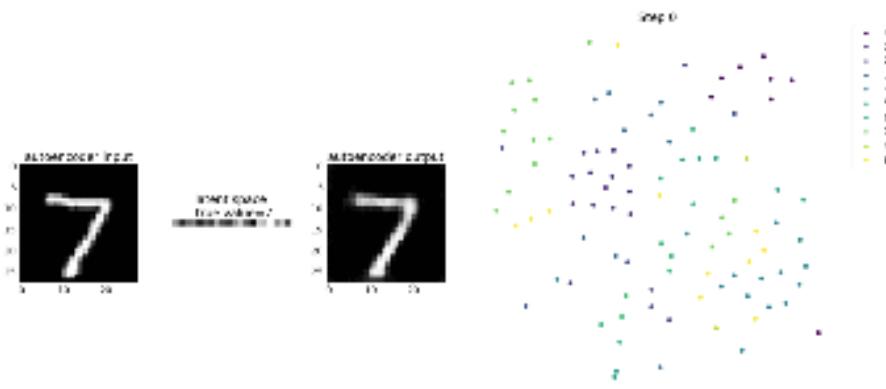
The training model in this notebook is a convolutional autoencoder with the MXNet framework. The convolutional autoencoder has a bottleneck-shaped convolutional neural network that consists of an encoder part and a decoder part.

The encoder in this example has two convolution layers to produce compressed representation (latent variables) of the input images. In this case, the encoder produces a latent variable of size (1, 20) from an original input image of size (28, 28) and significantly reduces the size of data for training by 40 times.

The decoder has two *deconvolutional* layers and ensures that the latent variables preserve key information by reconstructing output images.

The convolutional encoder powers clustering algorithms with smaller input data size and the performance of clustering algorithms such as k-means, k-NN, and t-Distributed Stochastic Neighbor Embedding (t-SNE).

This notebook example demonstrates how to visualize the latent variables using Debugger, as shown in the following animation. It also demonstrates how the t-SNE algorithm classifies the latent variables into ten clusters and projects them into a two-dimensional space. The scatter plot color scheme on the right side of the image reflects the true values to show how well the BERT model and t-SNE algorithm organize the latent variables into the clusters.



## Using SageMaker Debugger to Monitor Attentions in BERT Model Training

Bidirectional Encode Representations from Transformers (BERT) is a language representation model. As the name of model reflects, the BERT model builds on *transfer learning* and the *Transformer model* for natural language processing (NLP).

The BERT model is pre-trained on unsupervised tasks such as predicting missing words in a sentence or predicting the next sentence that naturally follows a previous sentence. The training data contains 3.3 billion words (tokens) of English text, from sources such as Wikipedia and electronic books. For a simple example, the BERT model can give a high *attention* to appropriate verb tokens or pronoun tokens from a subject token.

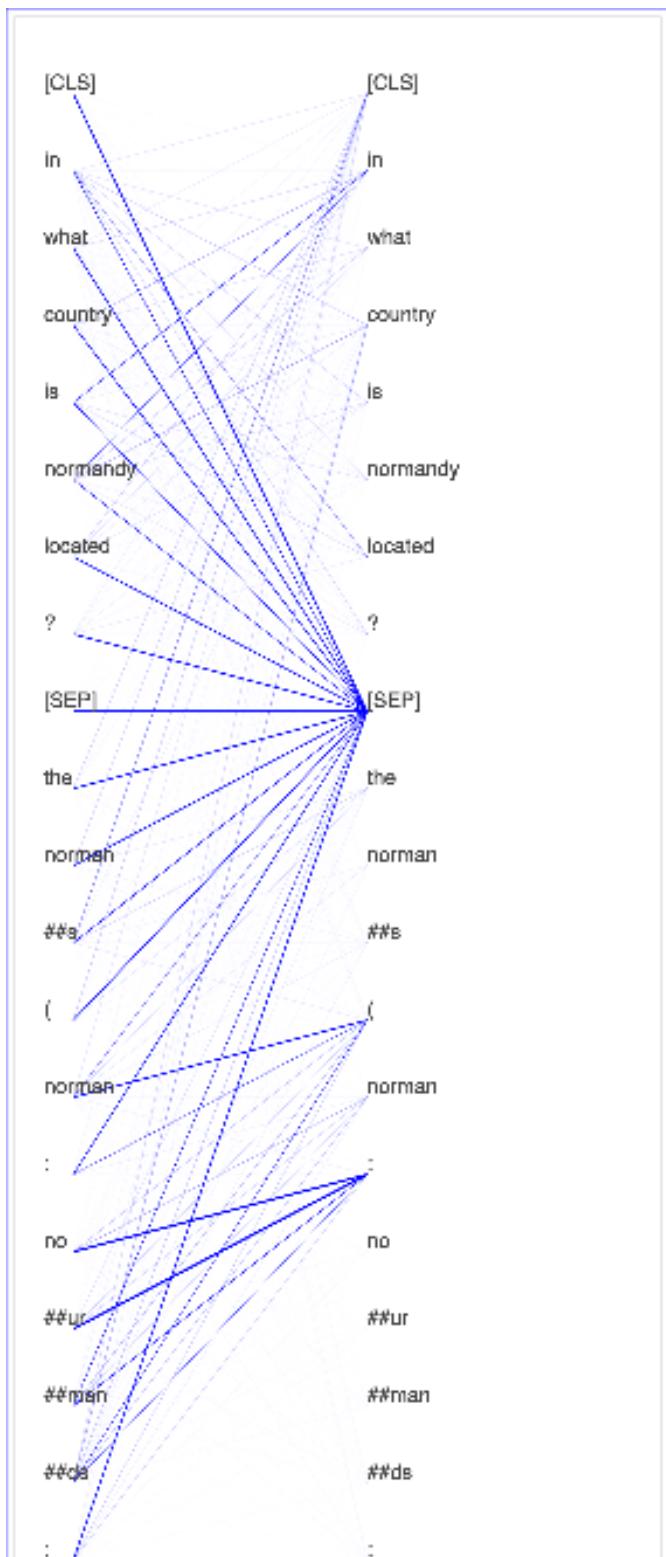
The pre-trained BERT model can be fine-tuned with an additional output layer to achieve state-of-the-art model training in NLP tasks, such as automated responses to questions, text classification, and many others.

Debugger collects tensors from the fine-tuning process. In the context of NLP, the weight of neurons is called *attention*.

This notebook demonstrates how to use the [pre-trained BERT model from the GluonNLP model zoo](#) on the Stanford Question and Answering dataset and how to set up SageMaker Debugger to monitor the training job.

Plotting *attention scores* and individual neurons in the query and key vectors can help to identify causes of incorrect model predictions. With SageMaker Debugger, you can retrieve the tensors and plot the *attention-head view* in real time as training progresses and understand what the model is learning.

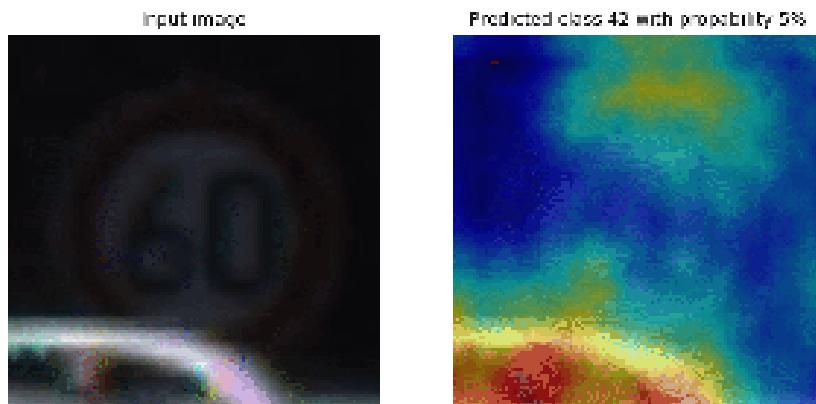
The following animation shows the attention scores of the first 20 input tokens for ten iterations in the training job provided in the notebook example.



## Using SageMaker Debugger to Visualize Class Activation Maps in Convolutional Neural Networks (CNNs)

This notebook demonstrates how to use SageMaker Debugger to plot class activation maps for image detection and classification in convolutional neural networks (CNNs). In deep learning, a *convolutional neural network (CNN or ConvNet)* is a class of deep neural networks, most commonly applied to analyzing visual imagery. One of the applications that adopts the class activation maps is self-driving cars, which require instantaneous detection and classification of images such as traffic signs, roads, and obstacles.

In this notebook, the PyTorch ResNet model is trained on [the German Traffic Sign Dataset](#), which contains more than 40 classes of traffic-related objects and more than 50,000 images in total.



During the training process, SageMaker Debugger collects tensors to plot the class activation maps in real time. As shown in the animated image, the class activation map (also called as a *saliency map*) highlights regions with high activation in red color.

Using tensors captured by Debugger, you can visualize how the activation map evolves during the model training. The model starts by detecting the edge on the lower-left corner at the beginning of the training job. As the training progresses, the focus shifts to the center and detects the speed limit sign, and the model successfully predicts the input image as Class 3, which is a class of speed limit 60km/h signs, with a 97% confidence level.

## Configure Debugger Using Amazon SageMaker Python SDK

To configure Debugger, use [Amazon SageMaker Python SDK](#) and specify Debugger-specific parameters while constructing SageMaker estimators. There are three parameters you need to configure: `profiler_config`, `debugger_hook_config`, and `rules`.

### Note

By default, Debugger monitors and debugs SageMaker training jobs without any Debugger-specific parameters configured in SageMaker estimators. Debugger collects system metrics every 500 milliseconds and basic output tensors (scalar outputs such as loss and accuracy) every 500 steps. It also runs the `ProfilerReport` rule to analyze the system metrics and aggregate the Studio Debugger insights dashboard and a profiling report. Debugger saves the output data in your secured S3 bucket.

### Important

To use the new Debugger features, you need to upgrade the SageMaker Python SDK and the SMDebug client library. In your iPython kernel, Jupyter Notebook, or JupyterLab environment, run the following code to install the latest versions of the libraries and restart the kernel.

```
import sys
import IPython
!{sys.executable} -m pip install -U sagemaker smdebug
IPython.Application.instance().kernel.do_shutdown(True)
```

## Construct a SageMaker Estimator with Debugger

The following example codes show how to construct a SageMaker estimator with the Debugger-specific parameters depending on a framework of your choice. Throughout the documentation in the following topics, you can find more information about how to configure the Debugger-specific parameters that you can mix and match as you want.

### Note

The following example codes are not directly executable. You need to proceed to the next sections and configure the Debugger-specific parameters.

### TensorFlow

To access the deep profiling feature for TensorFlow, currently you need to specify the latest Amazon deep learning container images with CUDA 11. For example, you must specify the specific image URI as shown in the following example code:

```
# An example of constructing a SageMaker TensorFlow estimator
import boto3
import sagemaker
from sagemaker.tensorflow import TensorFlow
from sagemaker.debugger import ProfilerConfig, DebuggerHookConfig, Rule, ProfilerRule,
    rule_configs

session=boto3.session.Session()
region=session.region_name

profiler_config=ProfilerConfig(...)
debugger_hook_config=DebuggerHookConfig(...)
rules=[
    Rule.sagemaker(rule_configs.built_in_rule()),
    ProfilerRule.sagemaker(rule_configs.BuiltInRule())
]

estimator=TensorFlow(
    entry_point="directory/to/your_training_script.py",
    role=sagemaker.get_execution_role(),
    base_job_name="debugger-demo",
    instance_count=1,
    instance_type="ml.p3.2xlarge",
    image_uri=f"763104351884.dkr.ecr.{region}.amazonaws.com/tensorflow-training:2.3.1-
gpu-py37-cu110-ubuntu18.04"

    # Debugger-specific parameters
    profiler_config=profiler_config,
    debugger_hook_config=debugger_hook_config,
    rules=rules
)

estimator.fit(wait=False)
```

## PyTorch

To access the deep profiling feature for PyTorch, currently you need to specify the latest Amazon deep learning container images with CUDA 11. For example, you must specify the specific image URI as shown in the following example code:

```
# An example of constructing a SageMaker PyTorch estimator
import boto3
import sagemaker
from sagemaker.pytorch import PyTorch
from sagemaker.debugger import ProfilerConfig, DebuggerHookConfig, Rule, ProfilerRule,
rule_configs

session=boto3.session.Session()
region=session.region_name

profiler_config=ProfilerConfig(...)
debugger_hook_config=DebuggerHookConfig(...)
rules=[
    Rule.sagemaker(rule_configs.built_in_rule()),
    ProfilerRule.sagemaker(rule_configs.BuiltInRule())
]

estimator=PyTorch(
    entry_point="directory/to/your_training_script.py",
    role=sagemaker.get_execution_role(),
    base_job_name="debugger-demo",
    instance_count=1,
    instance_type="ml.p3.2xlarge",
    image_uri=f"763104351884.dkr.ecr.{region}.amazonaws.com/pytorch-training:1.6.0-gpu-
py36-cu110-ubuntu18.04",

    # Debugger-specific parameters
    profiler_config=profiler_config,
    debugger_hook_config=debugger_hook_config,
    rules=rules
)

estimator.fit(wait=False)
```

## MXNet

```
# An example of constructing a SageMaker MXNet estimator
import sagemaker
from sagemaker.mxnet import MXNet
from sagemaker.debugger import ProfilerConfig, DebuggerHookConfig, Rule, ProfilerRule,
rule_configs

profiler_config=ProfilerConfig(...)
debugger_hook_config=DebuggerHookConfig(...)
rules=[
    Rule.sagemaker(rule_configs.built_in_rule()),
    ProfilerRule.sagemaker(rule_configs.BuiltInRule())
]

estimator=MXNet(
    entry_point="directory/to/your_training_script.py",
    role=sagemaker.get_execution_role(),
    base_job_name="debugger-demo",
    instance_count=1,
    instance_type="ml.p3.2xlarge",
    framework_version="1.7.0",
    py_version="py37",
```

```

        # Debugger-specific parameters
        profiler_config=profiler_config,
        debugger_hook_config=debugger_hook_config,
        rules=rules
    )

estimator.fit(wait=False)

```

#### **Note**

For MXNet, when configuring the `profiler_config` parameter, you can only configure for system monitoring. Profiling framework metrics is not supported for MXNet.

#### XGBoost

```

# An example of constructing a SageMaker XGBoost estimator
import sagemaker
from sagemaker.xgboost.estimator import XGBoost
from sagemaker.debugger import ProfilerConfig, DebuggerHookConfig, Rule, ProfilerRule,
rule_configs

profiler_config=ProfilerConfig(...)
debugger_hook_config=DebuggerHookConfig(...)
rules=[
    Rule.sagemaker(rule_configs.built_in_rule()),
    ProfilerRule.sagemaker(rule_configs.BuiltInRule())
]

estimator=XGBoost(
    entry_point="directory/to/your_training_script.py",
    role=sagemaker.get_execution_role(),
    base_job_name="debugger-demo",
    instance_count=1,
    instance_type="ml.p3.2xlarge",
    framework_version="1.2-1",

    # Debugger-specific parameters
    profiler_config=profiler_config,
    debugger_hook_config=debugger_hook_config,
    rules=rules
)

estimator.fit(wait=False)

```

#### **Note**

For XGBoost, when configuring the `profiler_config` parameter, you can only configure for system monitoring. Profiling framework metrics is not supported for XGBoost.

#### Generic estimator

```

# An example of constructing a SageMaker generic estimator using the XGBoost algorithm
base_image
import boto3
import sagemaker
from sagemaker.estimator import Estimator
from sagemaker import image_uris
from sagemaker.debugger import ProfilerConfig, DebuggerHookConfig, Rule, ProfilerRule,
rule_configs

profiler_config=ProfilerConfig(...)
debugger_hook_config=DebuggerHookConfig(...)
rules=[
    Rule.sagemaker(rule_configs.built_in_rule()),
    ProfilerRule.sagemaker(rule_configs.BuiltInRule())
]

```

```
]

region=boto3.Session().region_name
xgboost_container=sagemaker.image_uris.retrieve("xgboost", region, "1.2-1")

estimator=Estimator(
    role=sagemaker.get_execution_role(),
    image_uri=xgboost_container,
    base_job_name="debugger-demo",
    instance_count=1,
    instance_type="ml.m5.2xlarge",

    # Debugger-specific parameters
    profiler_config=profiler_config,
    debugger_hook_config=debugger_hook_config,
    rules=rules
)
estimator.fit(wait=False)
```

Where you configure the following parameters:

- **profiler\_config** parameter – Configure Debugger to collect system metrics and framework metrics from your training job and save into your secured S3 bucket URI or local machine. To learn how to configure the **profiler\_config** parameter, see [Configure Debugger Monitoring Hardware System Resource Utilization \(p. 1598\)](#) and [Configure Debugger Framework Profiling \(p. 1598\)](#).
- **debugger\_hook\_config** parameter – Configure Debugger to collect output tensors from your training job and save into your secured S3 bucket URI or local machine. To learn how to configure the **debugger\_hook\_config** parameter, see [Configure Debugger Hook to Save Tensors \(p. 1602\)](#).
- **rules** parameter – Configure this parameter to enable Debugger built-in rules that you want to run in parallel. The rules automatically analyze your training job and find training issues. The ProfilerReport rule saves the Debugger profiling reports in your secured S3 bucket URI. To learn how to configure the **rules** parameter, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

#### Note

Debugger securely saves output data in subfolders of your default S3 bucket. For example, the format of the default S3 bucket URI is `s3://sagemaker-<region>-<12digit_account_id>/<base-job-name>/<debugger-subfolders>/`. There are three subfolders created by Debugger: `debug-output`, `profiler-output`, and `rule-output`. You can also retrieve the default S3 bucket URLs using the [SageMaker estimator classmethods \(p. 1614\)](#).

See the following topics to find out how to configure the Debugger-specific parameters in detail.

#### Topics

- [Configure Debugger Monitoring Hardware System Resource Utilization \(p. 1598\)](#)
- [Configure Debugger Framework Profiling \(p. 1598\)](#)
- [Updating Debugger System Monitoring and Framework Profiling Configuration while a Training Job is Running \(p. 1602\)](#)
- [Configure Debugger Hook to Save Tensors \(p. 1602\)](#)
- [Configure Debugger Built-in Rules \(p. 1608\)](#)
- [Turn Off Debugger \(p. 1614\)](#)
- [Useful SageMaker Estimator Classmethods for Debugger \(p. 1614\)](#)

## Configure Debugger Monitoring Hardware System Resource Utilization

To adjust Debugger system monitoring time intervals, use the `ProfilerConfig` API operation to create a parameter object while constructing a SageMaker framework or generic estimator depending on your preference.

### Note

By default, for all SageMaker training jobs, Debugger collects hardware system utilization data from Amazon EC2 instances every 500 milliseconds for system monitoring, without any Debugger-specific parameters specified in SageMaker estimators.

Debugger saves the system metrics in a default S3 bucket. The format of the default S3 bucket URI is `s3://sagemaker-<region>-<12digit_account_id>/<training-job-name>/profiler-output/`.

The following example code shows how to set up the `profiler_config` parameter with a system monitoring time interval of 1000 milliseconds.

```
from sagemaker.debugger import ProfilerConfig

profiler_config=ProfilerConfig(
    system_monitor_interval_millis=1000
)
```

- `system_monitor_interval_millis` (int) – Specify the monitoring intervals in milliseconds to record system metrics. Available values are 100, 200, 500, 1000 (1 second), 5000 (5 seconds), and 60000 (1 minute) milliseconds. The default value is 500 milliseconds.

To see the progress of system monitoring, see [Open Amazon SageMaker Debugger Insights Dashboard \(p. 1686\)](#).

## Configure Debugger Framework Profiling

To enable Debugger framework profiling, configure the `framework_profile_params` parameter when you construct an estimator. Debugger framework profiling collects framework metrics, such as data from initialization stage, data loader processes, Python operators of deep learning frameworks and training scripts, detailed profiling within and between steps, with cProfile or Pyinstrument options. Using the `FrameworkProfile` class, you can configure custom framework profiling options.

### Note

Before getting started with Debugger framework profiling, verify that the framework used to build your model is supported by Debugger for framework profiling. For more information, see [Supported Frameworks and Algorithms \(p. 1581\)](#).

Debugger saves the framework metrics in a default S3 bucket. The format of the default S3 bucket URI is `s3://sagemaker-<region>-<12digit_account_id>/<training-job-name>/profiler-output/`.

## Start a Training Job with the Default System Monitoring and Framework Profiling

The following example code is the simplest `profiler_config` parameter setting to start the default system monitoring and the default framework profiling. The `FrameworkProfile` class in the following example code initiates the default framework profiling when a training job starts. Debugger framework profiling includes the following options: detailed profiling, data loader profiling, and Python profiling.

```
from sagemaker.debugger import ProfilerConfig, FrameworkProfile
```

```
profiler_config=ProfilerConfig(  
    framework_profile_params=FrameworkProfile()  
)
```

With this `profiler_config` parameter configuration, Debugger calls the default settings of monitoring and profiling. Debugger monitors system metrics every 500 milliseconds; profiles the fifth step with the detailed profiling option; the seventh step with the data loader profiling option; and the ninth, tenth, and eleventh steps with the Python profiling option.

To find available profiling configuration options, the default parameter settings, and examples of how to configure them, see [Start a Training Job with the Default System Monitoring and Customized Framework Profiling with Different Profiling Options \(p. 1600\)](#) and [SageMaker Debugger APIs – FrameworkProfile](#) in the [Amazon SageMaker Python SDK](#).

If you want to change the system monitoring interval and enable the default framework profiling, you can specify the `system_monitor_interval_millis` parameter explicitly with the `framework_profile_params` parameter. For example, to monitor every 1000 milliseconds and enable the default framework profiling, use the following example code.

```
from sagemaker.debugger import ProfilerConfig, FrameworkProfile  
  
profiler_config=ProfilerConfig(  
    system_monitor_interval_millis=1000,  
    framework_profile_params=FrameworkProfile()  
)
```

For more information about the `FrameworkProfile` class, see [SageMaker Debugger APIs – FrameworkProfile](#) in the [Amazon SageMaker Python SDK](#).

## Start a Training Job with the Default System Monitoring and Customized Framework Profiling for Target Steps or a Target Time Range

If you want to specify target steps or target time intervals to profile your training job, you need to specify parameters for the `FrameworkProfile` class. The following code examples show how to specify the target ranges for profiling along with system monitoring.

- **For a target step range**

With the following example configuration, Debugger monitors the entire training job every 500 milliseconds (the default monitoring) and profiles a target step range from step 5 to step 15 (for 10 steps).

```
from sagemaker.debugger import ProfilerConfig, FrameworkProfile  
  
profiler_config=ProfilerConfig(  
    framework_profile_params=FrameworkProfile(start_step=5, num_steps=10)  
)
```

With the following example configuration, Debugger monitors the entire training job every 1000 milliseconds and profiles a target step range from step 5 to step 15 (for 10 steps).

```
from sagemaker.debugger import ProfilerConfig, FrameworkProfile  
  
profiler_config=ProfilerConfig(  
    system_monitor_interval_millis=1000,  
    framework_profile_params=FrameworkProfile(start_step=5, num_steps=10)  
)
```

- **For a target time range**

With the following example configuration, Debugger monitors the entire training job every 500 milliseconds (the default monitoring) and profiles a target time range from the current Unix time for 600 seconds.

```
import time
from sagemaker.debugger import ProfilerConfig, FrameworkProfile

profiler_config=ProfilerConfig(
    framework_profile_params=FrameworkProfile(start_unix_time=int(time.time()),
                                                duration=600)
)
```

With the following example configuration, Debugger monitors the entire training job every 1000 milliseconds and profiles a target time range from the current Unix time for 600 seconds.

```
import time
from sagemaker.debugger import ProfilerConfig, FrameworkProfile

profiler_config=ProfilerConfig(
    system_monitor_interval_millis=1000,
    framework_profile_params=FrameworkProfile(start_unix_time=int(time.time()),
                                                duration=600)
)
```

The framework profiling is performed for all of the profiling options at the target step or time range.

To find more information about available profiling options, see [SageMaker Debugger APIs – FrameworkProfile](#) in the [Amazon SageMaker Python SDK](#).

The next section shows you how to script the available profiling options.

## Start a Training Job with the Default System Monitoring and Customized Framework Profiling with Different Profiling Options

You can use the following profiling configuration classes to manage the framework profiling options:

- [DetailedProfilingConfig](#) – Specify a target step or time range to profile framework operations using the native framework profilers (TensorFlow profiler and PyTorch profiler). For example, if using TensorFlow, the Debugger hooks enable the TensorFlow profiler to collect TensorFlow-specific framework metrics. Detailed profiling enables you to profile all framework operators at a pre-step (before the first step), within steps, and between steps of a training job.

**Note**

Detailed profiling might significantly increase GPU memory consumption. We do not recommend enabling detailed profiling for more than a couple of steps.

- [DataloaderProfilingConfig](#) – Specify a target step or time range to profile deep learning framework data loader processes. Debugger collects every data loader event of the frameworks.

**Note**

Data loader profiling might lower the training performance while collecting information from data loaders. We don't recommend enabling data loader profiling for more than a couple of steps.

Debugger is preconfigured to annotate data loader processes only for the Amazon deep learning containers. Debugger cannot profile data loader processes from any other custom or external training containers.

- [PythonProfilingConfig](#) – Specify a target step or time range to profile Python functions. You can also choose between two Python profilers: cProfile and Pyinstrument.

- *cProfile* – The standard Python profiler. *cProfile* collects information for every Python operator called during training. With *cProfile*, Debugger saves cumulative time and annotation for each function call, providing complete detail about Python functions. In deep learning, for example, the most frequently called functions might be the convolutional filters and backward pass operators, and *cProfile* profiles every single of them. For the *cProfile* option, you can further select a timer option: total time, CPU time, and off-CPU time. While you can profile every function call executing on processors (both CPU and GPU) in CPU time, you can also identify I/O or network bottlenecks with the off-CPU time option. The default is total time, and Debugger profiles both CPU and off-CPU time. With *cProfile*, you are able to drill down to every single functions when analyzing the profile data.
- *Pyinstrument* – *Pyinstrument* is a low-overhead Python profiler that works based on sampling. With the *Pyinstrument* option, Debugger samples profiling events every millisecond. Because *Pyinstrument* measures elapsed wall-clock time instead of CPU time, the *Pyinstrument* option can be a better choice over the *cProfile* option for reducing profiling noise (filtering out irrelevant function calls that are cumulatively fast) and capturing operators that are actually compute intensive (cumulatively slow) for training your model. With *Pyinstrument*, you are able to see a tree of function calls and better understand the structure and root cause of the slowness.

**Note**

Enabling Python profiling might slow down the overall training time. *cProfile* profiles the most frequently called Python operators at every call, so the processing time on profiling increases with respect to the number of calls. For *Pyinstrument*, the cumulative profiling time increases with respect to time because of its sampling mechanism.

The following example configuration shows the full structure when you use the different profiling options with specified values.

```
import time
from sagemaker.debugger import (ProfilerConfig,
                                 FrameworkProfile,
                                 DetailedProfilingConfig,
                                 DataloaderProfilingConfig,
                                 PythonProfilingConfig,
                                 PythonProfiler, cProfileTimer)

profiler_config=ProfilerConfig(
    system_monitor_interval_millis=500,
    framework_profile_params=FrameworkProfile(
        detailed_profiling_config=DetailedProfilingConfig(
            start_step=5,
            num_steps=1
        ),
        dataloader_profiling_config=DataloaderProfilingConfig(
            start_step=7,
            num_steps=1
        ),
        python_profiling_config=PythonProfilingConfig(
            start_step=9,
            num_steps=1,
            python_profiler=PythonProfiler.CPROFILE,
            cprofile_timer=cProfileTimer.TOTAL_TIME
        )
    )
)
```

For more information about available profiling options, see [DetailedProfilingConfig](#), [DataloaderProfilingConfig](#), and [PythonProfilingConfig](#) in the [Amazon SageMaker Python SDK](#).

## Updating Debugger System Monitoring and Framework Profiling Configuration while a Training Job is Running

If you want to enable or update the Debugger monitoring and profiling configuration for a training job that is currently running, use the following SageMaker estimator extension methods:

- To enable Debugger system monitoring for a running training job and receive a Debugger profiling report, use the following:

```
estimator.enable_default_profiling()
```

When you use the `enable_default_profiling` method, Debugger initiates the default system monitoring and the `ProfileReport` built-in rule, which generates a comprehensive profiling report at the end of the training job. This method can be called only if the current training job is running without both Debugger monitoring and profiling.

For more information, see [estimator.enable\\_default\\_profiling](#) in the [Amazon SageMaker Python SDK](#).

- To enable Debugger built-in rules, system monitoring, and framework profiling with customizable configuration parameters, use the following:

```
estimator.update_profiler(  
    rules=[ProfilerRule.sagemaker(rule_configs.BuiltInRule()),  
          system_monitor_interval_millis=500,  
          framework_profile_params=FrameworkProfile()  
)
```

For more information, see [estimator.update\\_profiler](#) in the [Amazon SageMaker Python SDK](#).

## Configure Debugger Hook to Save Tensors

*Tensors* are data collections of updated parameters from the backward and forward pass of each training iteration, and Debugger collects output tensors to analyze the state of a training job. The Amazon SageMaker Debugger `CollectionConfig` and `DebuggerHookConfig` API operations provide methods for grouping tensors into *collections* and saving them to a target S3 bucket.

### Note

By default, for all SageMaker training jobs, Debugger collects loss and accuracy output scalars from the training jobs every 500 steps, without any Debugger-specific parameters specified in SageMaker estimators. Debugger saves the output data in a default S3 bucket. The format of the default S3 bucket URI is `s3://sagemaker-<region>-<12digit_account_id>/<training-job-name>/debug-output/`.

While constructing a SageMaker estimator, enable Debugger by specifying the hook configuration parameter, `debugger_hook_config`. The following steps include examples of how to set up the `debugger_hook_config` using the `CollectionConfig` and `DebuggerHookConfig` API operations to pull tensors out of your training jobs and save them. If you use [Debugger-supported Amazon containers for zero script change](#), you can simply run the training job without changing your training script. You can also use Debugger for training jobs running in any other [Debugger-supported Amazon containers with script mode](#), making minimal changes to your training script.

### Configure Debugger Tensor Collections Using the CollectionConfig API Operation

Use the `CollectionConfig` API operation to configure tensor collections. Debugger provides pre-built tensor collections that cover a variety of regular expressions (regex) of parameters if using Debugger-

supported deep learning frameworks and machine learning algorithms. As shown in the following example code, add the built-in tensor collections you want to debug.

```
from sagemaker.debugger import CollectionConfig

collection_configs=[
    CollectionConfig(name="weights"),
    CollectionConfig(name="gradients")
]
```

The preceding collections set up the Debugger hook to save the tensors every 500 steps based on the default "save\_interval" value.

For a full list of available Debugger built-in collections, see [Debugger Built-in Collections](#).

If you want to customize the built-in collections, such as changing the save intervals and tensor regex, use the following `CollectionConfig` template to adjust parameters.

```
from sagemaker.debugger import CollectionConfig

collection_configs=[
    CollectionConfig(
        name="tensor_collection",
        parameters={
            "key_1": "value_1",
            "key_2": "value_2",
            ...
            "key_n": "value_n"
        }
    )
]
```

For more information about available parameter keys, see [CollectionConfig](#) in the [Amazon SageMaker Python SDK](#). For example, the following code example shows how you can adjust the save intervals of the "losses" tensor collection at different phases of training: save loss every 100 steps in training phase and validation loss every 10 steps in validation phase.

```
from sagemaker.debugger import CollectionConfig

collection_configs=[
    CollectionConfig(
        name="losses",
        parameters={
            "train.save_interval": "100",
            "eval.save_interval": "10"
        }
    )
]
```

**Tip**

This tensor collection configuration object can be used for both [DebuggerHookConfig](#) and [Rule](#) API operations.

## Configure Debugger Hook to Save Tensors

Use the [DebuggerHookConfig](#) class to create a `debugger_hook_config` object using the `collection_configs` object you created in the previous step.

```
from sagemaker.debugger import DebuggerHookConfig
```

```
    debugger_hook_config=DebuggerHookConfig(  
        collection_configs=collection_configs  
)
```

Debugger saves the model training output tensors into the default S3 bucket. The format of the default S3 bucket URI is `s3://sagemaker-<region>-<12digit_account_id>/<training-job-name>/debug-output/`.

If you want to specify an exact S3 bucket URI, use the following code example:

```
from sagemaker.debugger import DebuggerHookConfig  
  
debugger_hook_config=DebuggerHookConfig(  
    s3_output_path="specify-your-s3-bucket-uri"  
    collection_configs=collection_configs  
)
```

For more information, see [DebuggerHookConfig](#) in the [Amazon SageMaker Python SDK](#).

## Example Notebooks and Code Samples to Configure Debugger Hook

The following sections provide notebooks and code examples of how to use Debugger hook to save, access, and visualize output tensors.

### Topics

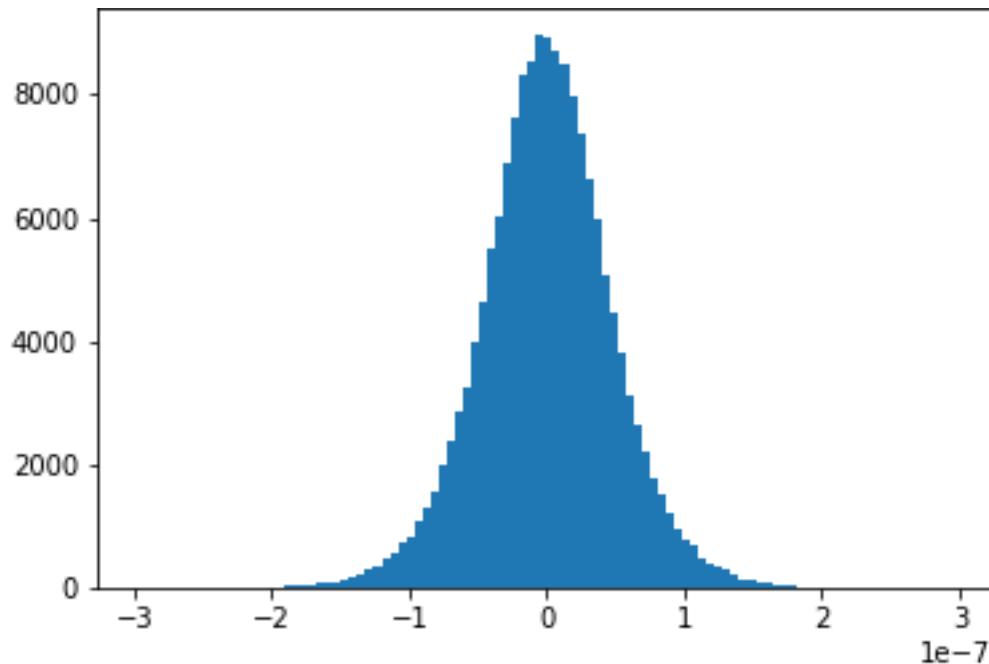
- [Tensor Visualization Example Notebooks \(p. 1604\)](#)
- [Save Tensors Using Debugger Built-in Collections \(p. 1606\)](#)
- [Save Tensors Using Debugger Modified Built-in Collections \(p. 1607\)](#)
- [Save Tensors Using Debugger Custom Collections \(p. 1607\)](#)

### Tensor Visualization Example Notebooks

The following two notebook examples show advanced use of Amazon SageMaker Debugger for visualizing tensors. Debugger provides a transparent view into training deep learning models.

- [Interactive Tensor Analysis in SageMaker Studio Notebook with MXNet](#)

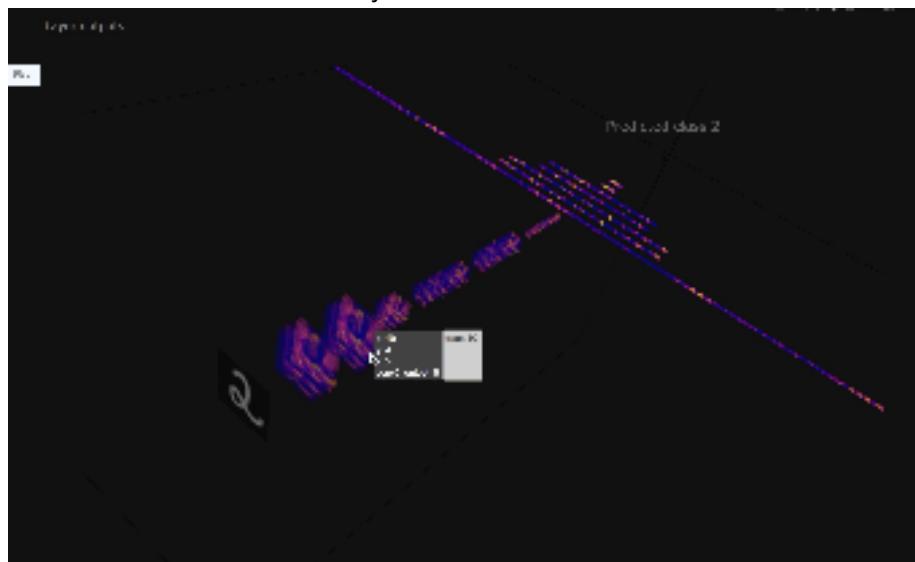
This notebook example shows how to visualize saved tensors using Amazon SageMaker Debugger. By visualizing the tensors, you can see how the tensor values change while training deep learning algorithms. This notebook includes a training job with a poorly configured neural network and uses Amazon SageMaker Debugger to aggregate and analyze tensors, including gradients, activation outputs, and weights. For example, the following plot shows the distribution of gradients of a convolutional layer that is suffering from a vanishing gradient problem.



This notebook also illustrates how a good initial hyperparameter setting improves the training process by generating the same tensor distribution plots.

- [Visualizing and Debugging Tensors from MXNet Model Training](#)

This notebook example shows how to save and visualize tensors from an MXNet Gluon model training job using Amazon SageMaker Debugger. It illustrates that Debugger is set to save all tensors to an Amazon S3 bucket and retrieves ReLU activation outputs for the visualization. The following figure shows a three-dimensional visualization of the ReLU activation outputs. The color scheme is set to blue to indicate values close to 0 and yellow to indicate values close to 1.



In this notebook, the `TensorPlot` class imported from `tensor_plot.py` is designed to plot convolutional neural networks (CNNs) that take two-dimensional images for inputs. The `tensor_plot.py` script provided with the notebook retrieves tensors using Debugger and visualizes

the CNN. You can run this notebook on SageMaker Studio to reproduce the tensor visualization and implement your own convolutional neural network model.

- [Real-time Tensor Analysis in a SageMaker Notebook with MXNet](#)

This example guides you through installing required components for emitting tensors in an Amazon SageMaker training job and using the Debugger API operations to access those tensors while training is running. A gluon CNN model is trained on the Fashion MNIST dataset. While the job is running, you will see how Debugger retrieves activation outputs of the first convolutional layer from each of 100 batches and visualizes them. Also, this will show you how to visualize weights after the job is done.

## Save Tensors Using Debugger Built-in Collections

You can use built-in collections of tensors using the `CollectionConfig` API and save them using the `DebuggerHookConfig` API. The following example shows how to use the default settings of Debugger hook configurations to construct a SageMaker TensorFlow estimator. You can also utilize this for MXNet, PyTorch, and XGBoost estimators.

### Note

In the following example code, the `s3_output_path` parameter for `DebuggerHookConfig` is optional. If you do not specify it, Debugger saves the tensors at `s3://<output_path>/debug-output/`, where the `<output_path>` is the default output path of SageMaker training jobs. For example:

```
"s3://sagemaker-us-east-1-111122223333/sagemaker-debugger-training-YYYY-MM-DD-HH-MM-SS-123/debug-output"
```

```
import sagemaker
from sagemaker.tensorflow import TensorFlow
from sagemaker.debugger import DebuggerHookConfig, CollectionConfig

# use Debugger CollectionConfig to call built-in collections
collection_configs=[
    CollectionConfig(name="weights"),
    CollectionConfig(name="gradients"),
    CollectionConfig(name="losses"),
    CollectionConfig(name="biases")
]

# configure Debugger hook
# set a target S3 bucket as you want
sagemaker_session=sagemaker.Session()
BUCKET_NAME=sagemaker_session.default_bucket()
LOCATION_IN_BUCKET='debugger-built-in-collections-hook'

hook_config=DebuggerHookConfig(
    s3_output_path='s3://{BUCKET_NAME}/{LOCATION_IN_BUCKET}'.
        format(BUCKET_NAME=BUCKET_NAME,
               LOCATION_IN_BUCKET=LOCATION_IN_BUCKET),
    collection_configs=collection_configs
)

# construct a SageMaker TensorFlow estimator
sagemaker_estimator=TensorFlow(
    entry_point='directory/to/your_training_script.py',
    role=sm.get_execution_role(),
    base_job_name='debugger-demo-job',
    instance_count=1,
    instance_type="ml.m4.xlarge",
    framework_version="2.3.0",
    py_version="py37",
```

```

        # debugger-specific hook argument below
        debugger_hook_config=hook_config
    )

sagemaker_estimator.fit()

```

To see a list of Debugger built-in collections, see [Debugger Built-in Collections](#).

### Save Tensors Using Debugger Modified Built-in Collections

You can modify the Debugger built-in collections using the `CollectionConfig` API operation. The following example shows how to tweak the built-in `losses` collection and construct a SageMaker TensorFlow estimator. You can also use this for MXNet, PyTorch, and XGBoost estimators.

```

import sagemaker
from sagemaker.tensorflow import TensorFlow
from sagemaker.debugger import DebuggerHookConfig, CollectionConfig

# use Debugger CollectionConfig to call and modify built-in collections
collection_configs=[
    CollectionConfig(
        name="losses",
        parameters={"save_interval": "50"})
]

# configure Debugger hook
# set a target S3 bucket as you want
sagemaker_session=sagemaker.Session()
BUCKET_NAME=sagemaker_session.default_bucket()
LOCATION_IN_BUCKET='debugger-modified-collections-hook'

hook_config=DebuggerHookConfig(
    s3_output_path='s3://{BUCKET_NAME}/{LOCATION_IN_BUCKET}'.format(BUCKET_NAME=BUCKET_NAME,
                                                                     LOCATION_IN_BUCKET=LOCATION_IN_BUCKET),
    collection_configs=collection_configs
)

# construct a SageMaker TensorFlow estimator
sagemaker_estimator=TensorFlow(
    entry_point='directory/to/your_training_script.py',
    role=sm.get_execution_role(),
    base_job_name='debugger-demo-job',
    instance_count=1,
    instance_type="ml.m4.xlarge",
    framework_version="2.3.0",
    py_version="py37",

    # debugger-specific hook argument below
    debugger_hook_config=hook_config
)

sagemaker_estimator.fit()

```

For a full list of `CollectionConfig` parameters, see [Debugger CollectionConfig API](#).

### Save Tensors Using Debugger Custom Collections

You can also save a reduced number of tensors instead of the full set of tensors (for example, if you want to reduce the amount of data saved in your Amazon S3 bucket). The following example shows how to customize the Debugger hook configuration to specify target tensors that you want to save. You can use this for TensorFlow, MXNet, PyTorch, and XGBoost estimators.

```

import sagemaker
from sagemaker.tensorflow import TensorFlow
from sagemaker.debugger import DebuggerHookConfig, CollectionConfig

# use Debugger CollectionConfig to create a custom collection
collection_configs=[
    CollectionConfig(
        name="custom_activations_collection",
        parameters={
            "include_regex": "relu|tanh", # Required
            "reductions": "mean,variance,max,abs_mean,abs_variance,abs_max"
        })
]

# configure Debugger hook
# set a target S3 bucket as you want
sagemaker_session=sagemaker.Session()
BUCKET_NAME=sagemaker_session.default_bucket()
LOCATION_IN_BUCKET='debugger-custom-collections-hook'

hook_config=DebuggerHookConfig(
    s3_output_path='s3://{}{}/{}'.format(BUCKET_NAME, LOCATION_IN_BUCKET),
    collection_configs=collection_configs
)

# construct a SageMaker TensorFlow estimator
sagemaker_estimator=TensorFlow(
    entry_point='directory/to/your_training_script.py',
    role=sm.get_execution_role(),
    base_job_name='debugger-demo-job',
    instance_count=1,
    instance_type="ml.m4.xlarge",
    framework_version="2.3.0",
    py_version="py37",

    # debugger-specific hook argument below
    debugger_hook_config=hook_config
)

sagemaker_estimator.fit()

```

For a full list of `CollectionConfig` parameters, see [Debugger CollectionConfig](#).

## Configure Debugger Built-in Rules

Amazon SageMaker Debugger rules analyze tensors emitted during the training of a model. Debugger offers the `Rule API` operation that monitors training job progress and errors for the success of training your model. For example, the rules can detect whether gradients are getting too large or too small, whether a model is overfitting or overtraining, and whether a training job does not decrease loss function and improve. To see a full list of available built-in rules, see [List of Debugger Built-in Rules \(p. 1626\)](#).

### Note

The built-in rules are prepared in Amazon SageMaker processing containers and fully managed by SageMaker Debugger. By default, Debugger initiates the [ProfilerReport \(p. 1628\)](#) rule for all SageMaker training jobs, without any Debugger-specific rule parameter specified to the SageMaker estimators. The `ProfilerReport` rule invokes all of the following built-in rules for monitoring system bottlenecks and profiling framework metrics:

- [BatchSize \(p. 1629\)](#)

- [CPUBottleneck \(p. 1631\)](#)
- [GPUMemoryIncrease \(p. 1632\)](#)
- [IOBottleneck \(p. 1633\)](#)
- [LoadBalancing \(p. 1634\)](#)
- [LowGPUUtilization \(p. 1635\)](#)
- [OverallSystemUsage \(p. 1636\)](#)
- [MaxInitializationTime \(p. 1636\)](#)
- [OverallFrameworkMetrics \(p. 1637\)](#)
- [StepOutlier \(p. 1637\)](#)

Debugger saves the profiling report in a default S3 bucket. The format of the default S3 bucket URI is `s3://sagemaker-<region>-<12digit_account_id>/<training-job-name>/rule-output/`. For more information about how to download the profiling report, see [SageMaker Debugger Profiling Report \(p. 1701\)](#). SageMaker Debugger fully manages the built-in rules and analyzes your training job in parallel. For more information about billing, see the **Amazon SageMaker Studio is available at no additional charge** section of the [Amazon SageMaker Pricing](#) page.

In the following topics, learn how to use the Debugger built-in rules.

### Topics

- [Use Debugger Built-in Rules with the Default Parameter Settings \(p. 1609\)](#)
- [Use Debugger Built-in Rules with Custom Parameter Values \(p. 1610\)](#)
- [Example Notebooks and Code Samples to Configure Debugger Rules \(p. 1611\)](#)

## Use Debugger Built-in Rules with the Default Parameter Settings

To specify Debugger built-in rules in an estimator, you need to configure a `rules` list object. The following example code shows the basic structure of listing the Debugger built-in rules:

```
from sagemaker.debugger import Rule, ProfilerRule, rule_configs

rules=[  
    ProfilerRule.sagemaker(rule_configs.BuiltInProfilerRuleName_1()),  
    ProfilerRule.sagemaker(rule_configs.BuiltInProfilerRuleName_2()),  
    ...  
    ProfilerRule.sagemaker(rule_configs.BuiltInProfilerRuleName_n()),  
    Rule.sagemaker(rule_configs.built_in_rule_name_1()),  
    Rule.sagemaker(rule_configs.built_in_rule_name_2()),  
    ...  
    Rule.sagemaker(rule_configs.built_in_rule_name_n())  
]
```

For more information about default parameter values and descriptions of the built-in rule, see [List of Debugger Built-in Rules \(p. 1626\)](#).

For example, to inspect the overall training performance and progress of your model, construct a SageMaker estimator with the following built-in rule configuration.

```
from sagemaker.debugger import Rule, rule_configs

rules=[
```

```

    ProfilerRule.sagemaker(rule_configs.ProfilerReport()),
    Rule.sagemaker(rule_configs.loss_not_decreasing()),
    Rule.sagemaker(rule_configs.overfit()),
    Rule.sagemaker(rule_configs.overtraining()),
    Rule.sagemaker(rule_configs.stalled_training_rule())
]

```

When you start the training job, Debugger collects system resource utilization data every 500 milliseconds and the loss and accuracy values every 500 steps by default. Debugger analyzes the resource utilization to identify if your model is having bottleneck problems. The `loss_not_decreasing`, `overfit`, `overtraining`, and `stalled_training_rule` monitors if your model is optimizing the loss function without those training issues. If the rules detect training anomalies, the rule evaluation status changes to `IssueFound`. You can set up automated actions, such as notifying training issues and stopping training jobs using Amazon CloudWatch Events and Amazon Lambda. For more information, see [Action on Amazon SageMaker Rules \(p. 1675\)](#).

## Use Debugger Built-in Rules with Custom Parameter Values

If you want to adjust the built-in rule parameter values and customize tensor collection regex, configure the `base_config` and `rule_parameters` parameters for the `ProfilerRule.sagemaker` and `Rule.sagemaker` classmethods. In case of the `Rule.sagemaker` class methods, you can also customize tensor collections through the `collections_to_save` parameter. The instruction of how to use the `CollectionConfig` class is provided at [Configure Debugger Tensor Collections Using the CollectionConfig API Operation \(p. 1602\)](#).

Use the following configuration template for built-in rules to customize parameter values. By changing the rule parameters as you want, you can adjust the sensitivity of the rules to be triggered.

- The `base_config` argument is where you call the built-in rule methods.
- The `rule_parameters` argument is to adjust the default key values of the built-in rules listed in [List of Debugger Built-in Rules \(p. 1626\)](#).
- The `collections_to_save` argument takes in a tensor configuration through the `CollectionConfig` API, which requires `name` and `parameters` arguments.
  - To find available tensor collections for `name`, see [Debugger Built-in Tensor Collections](#).
  - For a full list of adjustable `parameters`, see [Debugger CollectionConfig API](#).

For more information about the Debugger rule class, methods, and parameters, see [SageMaker Debugger Rule class](#) in the [Amazon SageMaker Python SDK](#).

```

from sagemaker.debugger import Rule, ProfilerRule, rule_configs, CollectionConfig

rules=[
    ProfilerRule.sagemaker(
        base_config=rule_configs.BuiltInProfilerRuleName(),
        rule_parameters={
            "key": "value"
        }
    )
    Rule.sagemaker(
        base_config=rule_configs.built_in_rule_name(),
        rule_parameters={
            "key": "value"
        }
    )
    collections_to_save=[
        CollectionConfig(
            name="tensor_collection_name",
            parameters={

```

```
        "key": "value"
    }
]
]
```

The parameter descriptions and value customization examples are provided for each rule at [List of Debugger Built-in Rules \(p. 1626\)](#).

## Example Notebooks and Code Samples to Configure Debugger Rules

In the following sections, notebooks and code samples of how to use Debugger rules to monitor SageMaker training jobs are provided.

### Topics

- [Debugger Built-in Rules Example Notebooks \(p. 1611\)](#)
- [Debugger Built-in Rules Example Code \(p. 1612\)](#)
- [Use Debugger Built-in Rules with Parameter Modifications \(p. 1613\)](#)

### Debugger Built-in Rules Example Notebooks

The following example notebooks show how to use Debugger built-in rules when running training jobs with Amazon SageMaker:

- [Using a SageMaker Debugger built-in rule with TensorFlow](#)
- [Using a SageMaker Debugger built-in rule with Managed Spot Training and MXNet](#)
- [Using a SageMaker Debugger built-in rule with XGBoost](#)
- [Using a SageMaker Debugger built-in rule with parameter modifications for a real-time training job analysis with XGBoost](#)

While running the example notebooks in SageMaker Studio, you can find the training job trial created on the **Studio Experiment List** tab. For example, as shown in the following screenshot, you can find and open a **Describe Trial Component** window of your current training job. On the Debugger tab, you can check if the Debugger rules, `vanishing_gradient()` and `loss_not_decreasing()`, are monitoring the training session in parallel. For a full instruction of how to find your training job trial components in the Studio UI, see [SageMaker Studio - View Experiments, Trials, and Trial Components](#).



There are two ways of using the Debugger built-in rules in the SageMaker environment: deploy the built-in rules as it is prepared or adjust their parameters as you want. The following topics show you how to use the built-in rules with example codes.

### Debugger Built-in Rules Example Code

The following code sample shows how to set the Debugger built-in rules using the `Rule.sagemaker` method. To specify built-in rules that you want to run, use the `rules_configs` API operation to call the built-in rules. To find a full list of Debugger built-in rules and default parameter values, see [List of Debugger Built-in Rules \(p. 1626\)](#).

```

import sagemaker
from sagemaker.tensorflow import TensorFlow
from sagemaker.debugger import Rule, CollectionConfig, rule_configs

# call built-in rules that you want to use.
built_in_rules=[
    Rule.sagemaker(rule_configs.vanishing_gradient())
    Rule.sagemaker(rule_configs.loss_not_decreasing())
]

# construct a SageMaker estimator with the Debugger built-in rules
sagemaker_estimator=TensorFlow(
    entry_point='directory/to/your_training_script.py',
    role=sm.get_execution_role(),
    base_job_name='debugger-built-in-rules-demo',
    rules=built_in_rules
)

```

```

    instance_count=1,
    instance_type="ml.m4.xlarge",
    framework_version="2.1.0",
    py_version="py3",

    # debugger-specific arguments below
    rules=built_in_rules
)
sagemaker_estimator.fit()

```

**Note**

The Debugger built-in rules run in parallel with your training job. The maximum number of built-in rule containers for a training job is 20.

For more information about the Debugger rule class, methods, and parameters, see the [SageMaker Debugger Rule class](#) in the [Amazon SageMaker Python SDK](#).

To find an example of how to adjust the Debugger rule parameters, see the following [Use Debugger Built-in Rules with Parameter Modifications \(p. 1613\)](#) section.

### Use Debugger Built-in Rules with Parameter Modifications

The following code example shows the structure of built-in rules to adjust parameters. In this example, the `stalled_training_rule` collects the `losses` tensor collection from a training job at every 50 steps and an evaluation stage at every 10 steps. If the training process starts stalling and not collecting tensor outputs for 120 seconds, the `stalled_training_rule` stops the training job.

```

import sagemaker
from sagemaker.tensorflow import TensorFlow
from sagemaker.debugger import Rule, CollectionConfig, rule_configs

# call the built-in rules and modify the CollectionConfig parameters

base_job_name_prefix= 'smdebug-stalled-demo-' + str(int(time.time()))

built_in_rules_modified=[

    Rule.sagemaker(
        base_config=rule_configs.stalled_training_rule(),
        rule_parameters={
            'threshold': '120',
            'training_job_name_prefix': base_job_name_prefix,
            'stop_training_on_fire' : 'True'
        }
    ),
    collections_to_save=[
        CollectionConfig(
            name="losses",
            parameters={
                "train.save_interval": "50"
                "eval.save_interval": "10"
            }
        )
    ]
]

# construct a SageMaker estimator with the modified Debugger built-in rule
sagemaker_estimator=TensorFlow(
    entry_point='directory/to/your_training_script.py',
    role=sm.get_execution_role(),
    base_job_name=base_job_name_prefix,
    instance_count=1,
    instance_type="ml.m4.xlarge",
    framework_version="2.1.0",
    py_version="py3",
)

```

```
# debugger-specific arguments below
rules=built_in_rules_modified
)
sagemaker_estimator.fit()
```

For an advanced configuration of the Debugger built-in rules using the [CreateTrainingJob API](#), see [Configure Debugger Using Amazon SageMaker API \(p. 1615\)](#).

## Turn Off Debugger

If you want to completely turn off Debugger, do one of the following:

- Before starting a training job, do the following:

To disable both monitoring and profiling, include the `disable_profiler` parameter to your estimator and set it to `True`.

**Warning**

If you disable it, you won't be able to view the comprehensive Studio Debugger insights dashboard and the autogenerated profiling report.

To disable debugging, set the `debugger_hook_config` parameter to `False`.

**Warning**

If you disable it, you won't be able to collect output tensors and cannot debug your model parameters.

```
estimator=Estimator(
    ...
    disable_profiler=True
    debugger_hook_config=False
)
```

For more information about the Debugger-specific parameters, see [SageMaker Estimator](#) in the [Amazon SageMaker Python SDK](#).

- While a training job is running, do the following:

To disable both monitoring and profiling while your training job is running, use the following estimator classmethod:

```
estimator.disable_profiling()
```

To disable framework profiling only and keep system monitoring, use the `update_profiler` method:

```
estimator.update_profiler(disable_framework_metrics=true)
```

For more information about the estimator extension methods, see the `estimator.disable_profiling` and `estimator.update_profiler` classmethods in the [Amazon SageMaker Python SDK](#) documentation.

## Useful SageMaker Estimator Classmethods for Debugger

The following estimator class methods are useful for accessing your SageMaker training job information and retrieving output paths of training data collected by Debugger. The following methods are executable after you initiate a training job with the `estimator.fit()` method.

- To check the base S3 bucket URI of a SageMaker training job:

```
estimator.output_path
```

- To check the base job name of a SageMaker training job:

```
estimator.latest_training_job.job_name
```

- To see a full CreateTrainingJob API operation configuration of a SageMaker training job:

```
estimator.latest_training_job.describe()
```

- To check a full list of the Debugger rules while a SageMaker training job is running:

```
estimator.latest_training_job.rule_job_summary()
```

- To check the S3 bucket URI where the model parameter data (output tensors) are saved:

```
estimator.latest_job_debugger_artifacts_path()
```

- To check the S3 bucket URI at where the model performance data (system and framework metrics) are saved:

```
estimator.latest_job_profiler_artifacts_path()
```

- To check the Debugger rule configuration for debugging output tensors:

```
estimator.debugger_rule_configs
```

- To check the list of the Debugger rules for debugging while a SageMaker training job is running:

```
estimator.debugger_rules
```

- To check the Debugger rule configuration for monitoring and profiling system and framework metrics:

```
estimator.profiler_rule_configs
```

- To check the list of the Debugger rules for monitoring and profiling while a SageMaker training job is running:

```
estimator.profiler_rules
```

For more information about the SageMaker estimator class and its methods, see [Estimator API](#) in the [Amazon SageMaker Python SDK](#).

## Configure Debugger Using Amazon SageMaker API

The preceding topics focus on using Debugger through Amazon SageMaker Python SDK, which is a wrapper around Amazon SDK for Python (Boto3) and SageMaker API operations. This offers a high-level experience of accessing the Amazon SageMaker API operations. In case you need to manually configure the SageMaker API operations using Amazon Boto3 or Amazon Command Line Interface (CLI) for other SDKs, such as Java, Go, and C++, this guide covers how to configure [CreateTrainingJob](#), [UpdateTrainingJob](#), Debugger configuration APIs, and their parameters to use the Debugger built-in and custom rules.

## Topics

- [JSON \(Amazon CLI\) \(p. 1616\)](#)
- [Amazon Boto3 \(p. 1620\)](#)

## JSON (Amazon CLI)

Amazon SageMaker Debugger built-in rules can be configured for a training job using the [DebugHookConfig](#), [DebugRuleConfiguration](#), [ProfilerConfig](#), and [ProfilerRuleConfiguration](#) objects through the SageMaker [CreateTrainingJob](#) API operation. You need to specify the right image URI in the `RuleEvaluatorImage` parameter, and the following examples walk you through how to set up the JSON strings to request [CreateTrainingJob](#).

The following code shows a complete JSON template to run a training job with required settings and Debugger configurations. Save the template as a JSON file in your working directory and run the training job using Amazon CLI. For example, save the following code as `debugger-training-job-cli.json`.

### Note

Ensure that you use the correct Docker container images. To find Amazon Deep Learning Container images, see [Available Deep Learning Containers Images](#). To find a complete list of available Docker images for using the Debugger rules, see [Use Debugger Docker Images for Built-in or Custom Rules \(p. 1738\)](#).

```
{
    "TrainingJobName": "debugger-aws-cli-test",
    "RoleArn": "arn:aws:iam::111122223333:role/service-role/AmazonSageMaker-ExecutionRole-YYYYMMDDT123456",
    "AlgorithmSpecification": {
        // Specify a training Docker container image URI (Deep Learning Container or your own training container) to TrainingImage.
        "TrainingImage": "763104351884.dkr.ecr.us-west-2.amazonaws.com/tensorflow-training:2.4.1-gpu-py37-cu110-ubuntu18.04",
        "TrainingInputMode": "File",
        "EnableSageMakerMetricsTimeSeries": false
    },
    "HyperParameters": {
        "sagemaker_program": "entry_point/tf-hvd-train.py",
        "sagemaker_submit_directory": "s3://sagemaker-us-west-2-111122223333/debugger-boto3-profiling-test/source.tar.gz"
    },
    "OutputDataConfig": {
        "S3OutputPath": "s3://sagemaker-us-west-2-111122223333/debugger-aws-cli-test/output"
    },
    "DebugHookConfig": {
        "S3OutputPath": "s3://sagemaker-us-west-2-111122223333/debugger-aws-cli-test/debug-output",
        "CollectionConfigurations": [
            {
                "CollectionName": "losses",
                "CollectionParameters": {
                    "train.save_interval": "50"
                }
            }
        ]
    },
    "DebugRuleConfigurations": [
        {
            "RuleConfigurationName": "LossNotDecreasing",
            "RuleEvaluatorImage": "895741380848.dkr.ecr.us-west-2.amazonaws.com/sagemaker-debugger-rules:latest",
            "RuleParameters": {"rule_to_invoke": "LossNotDecreasing"}
        }
    ]
}
```

```

        ],
        "ProfilerConfig": {
            "S3OutputPath": "s3://sagemaker-us-west-2-111122223333/debugger-aws-cli-test/
profiler-output",
            "ProfilingIntervalInMilliseconds": 500,
            "ProfilingParameters": {
                "DataloaderProfilingConfig": "{\"StartStep\": 5, \"NumSteps\": 3, \"MetricsRegex
\": \".*\", }",
                "DetailedProfilingConfig": "{\"StartStep\": 5, \"NumSteps\": 3, }",
                "PythonProfilingConfig": "{\"StartStep\": 5, \"NumSteps\": 3, \"ProfilerName\":
\"cprofile\", \"cProfileTimer\": \"total_time\"}",
                "LocalPath": "/opt/ml/output/profiler/"
            }
        },
        "ProfilerRuleConfigurations": [
            {
                "RuleConfigurationName": "ProfilerReport",
                "RuleEvaluatorImage": "895741380848.dkr.ecr.us-west-2.amazonaws.com/sagemaker-
debugger-rules:latest",
                "RuleParameters": {"rule_to_invoke": "ProfilerReport"}
            }
        ],
        "ResourceConfig": {
            "InstanceType": "ml.p3.8xlarge",
            "InstanceCount": 1,
            "VolumeSizeInGB": 30
        },
        "StoppingCondition": {
            "MaxRuntimeInSeconds": 86400
        }
    }
}

```

After saving the JSON file, run the following command in your terminal. (Use ! at the beginning of the line if you use a Jupyter notebook.)

```
aws sagemaker create-training-job --cli-input-json file://debugger-training-job-cli.json
```

## To configure a Debugger rule for debugging model parameters

The following code samples show how to configure a built-in VanishingGradient rule using this SageMaker API.

### To enable Debugger to collect output tensors

Specify the Debugger hook configuration as follows:

```

"DebugHookConfig": {
    "S3OutputPath": "s3://<default-bucket>/<training-job-name>/debug-output",
    "CollectionConfigurations": [
        {
            "CollectionName": "gradients",
            "CollectionParameters" : {
                "save_interval": "500"
            }
        }
    ]
}

```

This will make the training job save the tensor collection, gradients, every save\_interval of 500 steps. To find available CollectionName values, see [Debugger Built-in Collections](#) in the *SMDebug*

*client library documentation.* To find available CollectionParameters parameter keys and values, see the `sagemaker.debugger.CollectionConfig` class in the *SageMaker Python SDK documentation*.

### To enable Debugger rules for debugging the output tensors

The following DebugRuleConfigurations API example shows how to run the built-in VanishingGradient rule on the saved gradients collection.

```
"DebugRuleConfigurations": [
    {
        "RuleConfigurationName": "VanishingGradient",
        "RuleEvaluatorImage": "503895931360.dkr.ecr.us-east-1.amazonaws.com/sagemaker-
debugger-rules:latest",
        "RuleParameters": {
            "rule_to_invoke": "VanishingGradient",
            "threshold": "20.0"
        }
    }
]
```

With a configuration like the one in this sample, Debugger starts a rule evaluation job for your training job using the VanishingGradient rule on the collection of gradients tensor. To find a complete list of available Docker images for using the Debugger rules, see [Use Debugger Docker Images for Built-in or Custom Rules \(p. 1738\)](#). To find the key-value pairs for RuleParameters, see [List of Debugger Built-in Rules \(p. 1626\)](#).

### To configure a Debugger built-in rule for profiling system and framework metrics

The following example code shows how to specify the ProfilerConfig API operation to enable collecting system and framework metrics.

#### To enable Debugger profiling to collect system and framework metrics

##### Target Step

```
"ProfilerConfig": {
    // Optional. Path to an S3 bucket to save profiling outputs
    "S3OutputPath": "s3://<default-bucket>/<training-job-name>/profiler-output",
    // Available values for ProfilingIntervalInMilliseconds: 100, 200, 500, 1000 (1
second), 5000 (5 seconds), and 60000 (1 minute) milliseconds.
    "ProfilingIntervalInMilliseconds": 500,
    "ProfilingParameters": {
        "DataloaderProfilingConfig": "{ \"StartStep\": 5, \"NumSteps\": 3,
\"MetricsRegex\": \".*\\\" }",
        "DetailedProfilingConfig": "{ \"StartStep\": 5, \"NumSteps\": 3 }",
        // For PythonProfilingConfig,
        // available ProfilerName options: cProfile, Pyinstrument
        // available cProfileTimer options only when using cProfile: cpu, off_cpu,
        total_time
        "PythonProfilingConfig": "{ \"StartStep\": 5, \"NumSteps\": 3, \"ProfilerName\":
\"cProfile\", \"cProfileTimer\": \"total_time\" }",
        // Optional. Local path for profiling outputs
        "LocalPath": "/opt/ml/output/profiler/"
    }
}
```

##### Target Time Duration

```
"ProfilerConfig": {
    // Optional. Path to an S3 bucket to save profiling outputs
```

```

    "S3OutputPath": "s3://<default-bucket>/<training-job-name>/profiler-output",
    // Available values for ProfilingIntervalInMilliseconds: 100, 200, 500, 1000 (1
second), 5000 (5 seconds), and 60000 (1 minute) milliseconds.
    "ProfilingIntervalInMilliseconds": 500,
    "ProfilingParameters": {
        "DataLoaderProfilingConfig": "{ \"StartTimeInSecSinceEpoch\": 12345567789,
\"DurationInSeconds\": 10, \"MetricsRegex\": \".*\" }",
        "DetailedProfilingConfig": "{ \"StartTimeInSecSinceEpoch\": 12345567789,
\"DurationInSeconds\": 10 }",
        // For PythonProfilingConfig,
        // available ProfilerName options: cProfile, Pyinstrument
        // available cProfileTimer options only when using cProfile: cpu, off_cpu,
total_time
        "PythonProfilingConfig": "{ \"StartTimeInSecSinceEpoch\": 12345567789,
\"DurationInSeconds\": 10, \"ProfilerName\": \"cProfile\", \"cProfileTimer\":
\"total_time\" }",
        // Optional. Local path for profiling outputs
        "LocalPath": "/opt/ml/output/profiler/"
    }
}

```

### To enable Debugger rules for profiling the metrics

The following example code shows how to configure the `ProfilerReport` rule.

```

"ProfilerRuleConfigurations": [
{
    "RuleConfigurationName": "ProfilerReport",
    "RuleEvaluatorImage": "895741380848.dkr.ecr.us-west-2.amazonaws.com/sagemaker-
debugger-rules:latest",
    "RuleParameters": {
        "rule_to_invoke": "ProfilerReport",
        "CPUBottleneck_cpu_threshold": "90",
        "IOBottleneck_threshold": "90"
    }
}
]

```

To find a complete list of available Docker images for using the Debugger rules, see [Use Debugger Docker Images for Built-in or Custom Rules \(p. 1738\)](#). To find the key-value pairs for `RuleParameters`, see [List of Debugger Built-in Rules \(p. 1626\)](#).

## Update Debugger Profiling Configuration Using the `UpdateTrainingJob` API Operation

Debugger profiling configuration can be updated while your training job is running by using the `UpdateTrainingJob` API operation. Configure new `ProfilerConfig` and `ProfilerRuleConfiguration` objects, and specify the training job name to the `TrainingJobName` parameter.

```

{
    "ProfilerConfig": {
        "DisableProfiler": boolean,
        "ProfilingIntervalInMilliseconds": number,
        "ProfilingParameters": {
            "string" : "string"
        }
    },
    "ProfilerRuleConfigurations": [
        {
            "RuleConfigurationName": "string",
            "RuleEvaluatorImage": "string",
            ...
        }
    ]
}

```

```

        "RuleParameters": {
            "string" : "string"
        }
    },
    "TrainingJobName": "your-training-job-name-YYYY-MM-DD-HH-MM-SS-SSS"
}

```

## Add Debugger Custom Rule Configuration to the CreateTrainingJob API Operation

A custom rule can be configured for a training job using the [DebugHookConfig](#) and [DebugRuleConfiguration](#) objects in the [CreateTrainingJob](#) API operation. The following code sample shows how to configure a custom `ImproperActivation` rule written with the `smdebug` library using this SageMaker API operation. This example assumes that you've written the custom rule in `custom_rules.py` file and uploaded it to an Amazon S3 bucket. The example provides pre-built Docker images that you can use to run your custom rules. These are listed at [Amazon SageMaker Debugger Registry URLs for Custom Rule Evaluators \(p. 1739\)](#). You specify the URL registry address for the pre-built Docker image in the `RuleEvaluatorImage` parameter.

```

"DebugHookConfig": {
    "S3OutputPath": "s3://<default-bucket>/<training-job-name>/debug-output",
    "CollectionConfigurations": [
        {
            "CollectionName": "relu_activations",
            "CollectionParameters": {
                "include_regex": "relu",
                "save_interval": "500",
                "end_step": "5000"
            }
        }
    ],
    "DebugRulesConfigurations": [
        {
            "RuleConfigurationName": "improper_activation_job",
            "RuleEvaluatorImage": "552407032007.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-
debugger-rule-evaluator:latest",
            "InstanceType": "ml.c4.xlarge",
            "VolumeSizeInGB": 400,
            "RuleParameters": {
                "source_s3_uxi": "s3://bucket/custom_rules.py",
                "rule_to_invoke": "ImproperActivation",
                "collection_names": "relu_activations"
            }
        }
    ]
}

```

To find a complete list of available Docker images for using the Debugger rules, see [Use Debugger Docker Images for Built-in or Custom Rules \(p. 1738\)](#). To find the key-value pairs for `RuleParameters`, see [List of Debugger Built-in Rules \(p. 1626\)](#).

## Amazon Boto3

Amazon SageMaker Debugger built-in rules can be configured for a training job using the `create_training_job()` function of the Amazon Boto3 SageMaker client. You need to specify the right image URI in the `RuleEvaluatorImage` parameter, and the following examples walk you through how to set up the request body for the `create_training_job()` function.

The following code shows a complete example of how to configure Debugger for the `create_training_job()` request body and start a training job in `us-west-2`, assuming that a

training script `entry_point/train.py` is prepared using TensorFlow. To find an end-to-end example notebook, see [Profiling TensorFlow Multi GPU Multi Node Training Job with Amazon SageMaker Debugger \(Boto3\)](#).

**Note**

Ensure that you use the correct Docker container images. To find available Amazon Deep Learning Container images, see [Available Deep Learning Containers Images](#). To find a complete list of available Docker images for using the Debugger rules, see [Use Debugger Docker Images for Built-in or Custom Rules \(p. 1738\)](#).

```
import sagemaker, boto3
import datetime, tarfile

# Start setting up a SageMaker session and a Boto3 SageMaker client
session = sagemaker.Session()
region = session.boto_region_name
bucket = session.default_bucket()

# Upload a training script to a default Amazon S3 bucket of the current SageMaker session
source = 'source.tar.gz'
project = 'debugger-boto3-test'

tar = tarfile.open(source, 'w:gz')
tar.add ('entry_point/train.py') # Specify the directory and name of your training script
tar.close()

s3 = boto3.client('s3')
s3.upload_file(source, bucket, project+'/'+source)

# Set up a Boto3 session client for SageMaker
sm = boto3.Session(region_name=region).client("sagemaker")

# Start a training job
sm.create_training_job(
    TrainingJobName='debugger-boto3-'+datetime.datetime.now().strftime('%Y-%m-%d-%H-%M-%S'),
    HyperParameters={
        'sagemaker_submit_directory': 's3://'+bucket+'/'+project+'/'+source,
        'sagemaker_program': '/entry_point/train.py' # training scrip file location and name under the sagemaker_submit_directory
    },
    AlgorithmSpecification={
        # Specify a training Docker container image URI (Deep Learning Container or your own training container) to TrainingImage.
        'TrainingImage': '763104351884.dkr.ecr.us-west-2.amazonaws.com/tensorflow-training:2.4.1-gpu-py37-cu110-ubuntu18.04',
        'TrainingInputMode': 'File',
        'EnableSageMakerMetricsTimeSeries': False
    },
    RoleArn='arn:aws:iam::111122223333:role/service-role/AmazonSageMaker-ExecutionRole-20201014T161125',
    OutputDataConfig={'S3OutputPath': 's3://'+bucket+'/'+project+'/output'},
    ResourceConfig={
        'InstanceType': 'ml.p3.8xlarge',
        'InstanceCount': 1,
        'VolumeSizeInGB': 30
    },
    StoppingCondition={
        'MaxRuntimeInSeconds': 86400
    },
    DebugHookConfig={
        'S3OutputPath': 's3://'+bucket+'/'+project+'/debug-output',
        'CollectionConfigurations': [
            {
                'CollectionName': 'losses',
            }
        ]
    }
)
```

```

        'CollectionParameters' : {
            'train.save_interval': '500',
            'eval.save_interval': '50'
        }
    }
},
DebugRuleConfigurations=[
{
    'RuleConfigurationName': 'LossNotDecreasing',
    'RuleEvaluatorImage': '895741380848.dkr.ecr.us-west-2.amazonaws.com/sagemaker-
debugger-rules:latest',
    'RuleParameters': {'rule_to_invoke': 'LossNotDecreasing'}
},
],
ProfilerConfig={
    'S3OutputPath': 's3://'+bucket+'/'+project+'/profiler-output',
    'ProfilingIntervalInMilliseconds': 500,
    'ProfilingParameters': {
        'DataloaderProfilingConfig': '{"StartStep": 5, "NumSteps": 3, "MetricsRegex": 
".*", }',
        'DetailedProfilingConfig': '{"StartStep": 5, "NumSteps": 3, }',
        'PythonProfilingConfig': '{"StartStep": 5, "NumSteps": 3, "ProfilerName": 
"cprofile", "cProfileTimer": "total_time"}',
        'LocalPath': '/opt/ml/output/profiler/' # Optional. Local path for profiling
outputs
    }
},
ProfilerRuleConfigurations=[
{
    'RuleConfigurationName': 'ProfilerReport',
    'RuleEvaluatorImage': '895741380848.dkr.ecr.us-west-2.amazonaws.com/sagemaker-
debugger-rules:latest',
    'RuleParameters': {'rule_to_invoke': 'ProfilerReport'}
}
]
)

```

To configure a Debugger rule for debugging model parameters

The following code samples show how to configure a built-in `VanishingGradient` rule using this SageMaker API.

## To enable Debugger to collect output tensors

Specify the Debugger hook configuration as follows:

```
DebugHookConfig={  
    'S3OutputPath': 's3://<default-bucket>/<training-job-name>/debug-output',  
    'CollectionConfigurations': [  
        {  
            'CollectionName': 'gradients',  
            'CollectionParameters' : {  
                'train.save_interval': '500',  
                'eval.save_interval': '50'  
            }  
        }  
    ]  
}
```

This will make the training job save a tensor collection, gradients, every `save_interval` of 500 steps. To find available `CollectionName` values, see [Debugger Built-in Collections](#) in the *SMDebug client*.

*library documentation.* To find available CollectionParameters parameter keys and values, see the [sagemaker.debugger.CollectionConfig](#) class in the *SageMaker Python SDK documentation*.

### To enable Debugger rules for debugging the output tensors

The following DebugRuleConfigurations API example shows how to run the built-in VanishingGradient rule on the saved gradients collection.

```
DebugRuleConfigurations=[
    {
        'RuleConfigurationName': 'VanishingGradient',
        'RuleEvaluatorImage': '895741380848.dkr.ecr.us-west-2.amazonaws.com/sagemaker-
debugger-rules:latest',
        'RuleParameters': {
            'rule_to_invoke': 'VanishingGradient',
            'threshold': '20.0'
        }
    }
]
```

With a configuration like the one in this sample, Debugger starts a rule evaluation job for your training job using the VanishingGradient rule on the collection of gradients tensor. To find a complete list of available Docker images for using the Debugger rules, see [Use Debugger Docker Images for Built-in or Custom Rules \(p. 1738\)](#). To find the key-value pairs for RuleParameters, see [List of Debugger Built-in Rules \(p. 1626\)](#).

### To configure a Debugger built-in rule for profiling system and framework metrics

The following example code shows how to specify the ProfilerConfig API operation to enable collecting system and framework metrics.

#### To enable Debugger profiling to collect system and framework metrics

##### Target Step

```
ProfilerConfig={
    'S3OutputPath': 's3://<default-bucket>/<training-job-name>/profiler-output', #
    Optional. Path to an S3 bucket to save profiling outputs
    # Available values for ProfilingIntervalInMilliseconds: 100, 200, 500, 1000 (1
    second), 5000 (5 seconds), and 60000 (1 minute) milliseconds.
    'ProfilingIntervalInMilliseconds': 500,
    'ProfilingParameters': {
        'DataloaderProfilingConfig': '{
            "StartStep": 5,
            "NumSteps": 3,
            "MetricsRegex": ".*"
        }',
        'DetailedProfilingConfig': '{
            "StartStep": 5,
            "NumSteps": 3
        }',
        'PythonProfilingConfig': '{
            "StartStep": 5,
            "NumSteps": 3,
            "ProfilerName": "cprofile", # Available options: cprofile, pyinstrument
            "cProfileTimer": "total_time" # Include only when using cprofile.
        }',
        'LocalPath': '/opt/ml/output/profiler/' # Optional. Local path for profiling
    outputs
    }
```

```
}
```

## Target Time Duration

```

ProfilerConfig={
    'S3OutputPath': 's3://<default-bucket>/<training-job-name>/profiler-output', #
    Optional. Path to an S3 bucket to save profiling outputs
    # Available values for ProfilingIntervalInMilliseconds: 100, 200, 500, 1000 (1
    # second), 5000 (5 seconds), and 60000 (1 minute) milliseconds.
    'ProfilingIntervalInMilliseconds': 500,
    'ProfilingParameters': {
        'DataloaderProfilingConfig': '{'
            "StartTimeInSecSinceEpoch": 12345567789,
            "DurationInSeconds": 10,
            "MetricsRegex": ".*"
        },
        'DetailedProfilingConfig': '{'
            "StartTimeInSecSinceEpoch": 12345567789,
            "DurationInSeconds": 10
        },
        'PythonProfilingConfig': '{'
            "StartTimeInSecSinceEpoch": 12345567789,
            "DurationInSeconds": 10,
            "ProfilerName": "cprofile", # Available options: cprofile, pyinstrument
            "cProfileTimer": "total_time" # Include only when using cprofile.
        Available options: cpu, off_cpu, total_time
        },
        'LocalPath': '/opt/ml/output/profiler/' # Optional. Local path for profiling
    outputs
    }
}

```

## To enable Debugger rules for profiling the metrics

The following example code shows how to configure the `ProfilerReport` rule.

```

ProfilerRuleConfigurations=[
    {
        'RuleConfigurationName': 'ProfilerReport',
        'RuleEvaluatorImage': '895741380848.dkr.ecr.us-west-2.amazonaws.com/sagemaker-
debugger-rules:latest',
        'RuleParameters': {
            'rule_to_invoke': 'ProfilerReport',
            'CPUBottleneck_cpu_threshold': '90',
            'IOBottleneck_threshold': '90'
        }
    }
]

```

To find a complete list of available Docker images for using the Debugger rules, see [Use Debugger Docker Images for Built-in or Custom Rules \(p. 1738\)](#). To find the key-value pairs for `RuleParameters`, see [List of Debugger Built-in Rules \(p. 1626\)](#).

## Update Debugger Profiling Configuration Using the `UpdateTrainingJob` API Operation

Debugger profiling configuration can be updated while your training job is running by using the `update_training_job()` function of the Amazon Boto3 SageMaker client. Configure new `ProfilerConfig` and `ProfilerRuleConfiguration` objects, and specify the training job name to the `TrainingJobName` parameter.

```

ProfilerConfig={
    'DisableProfiler': boolean,
    'ProfilingIntervalInMilliseconds': number,
    'ProfilingParameters': {
        'string' : 'string'
    }
},
ProfilerRuleConfigurations=[
{
    'RuleConfigurationName': 'string',
    'RuleEvaluatorImage': 'string',
    'RuleParameters': {
        'string' : 'string'
    }
}
],
TrainingJobName='your-training-job-name-YYYY-MM-DD-HH-MM-SS-SSS'

```

## Add Debugger Custom Rule Configuration to the CreateTrainingJob API Operation

A custom rule can be configured for a training job using the [DebugHookConfig](#) and [DebugRuleConfiguration](#) objects using the Amazon Boto3 SageMaker client's [create\\_training\\_job\(\)](#) function. The following code sample shows how to configure a custom `ImproperActivation` rule written with the `smdebug` library using this SageMaker API operation. This example assumes that you've written the custom rule in `custom_rules.py` file and uploaded it to an Amazon S3 bucket. The example provides pre-built Docker images that you can use to run your custom rules. These are listed at [Amazon SageMaker Debugger Registry URLs for Custom Rule Evaluators \(p. 1739\)](#). You specify the URL registry address for the pre-built Docker image in the `RuleEvaluatorImage` parameter.

```

DebugHookConfig={
    'S3OutputPath': 's3://<default-bucket>/<training-job-name>/debug-output',
    'CollectionConfigurations': [
        {
            'CollectionName': 'relu_activations',
            'CollectionParameters': {
                'include_regex': 'relu',
                'save_interval': '500',
                'end_step': '5000'
            }
        }
    ]
},
DebugRulesConfigurations=[
{
    'RuleConfigurationName': 'improper_activation_job',
    'RuleEvaluatorImage': '552407032007.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-
debugger-rule-evaluator:latest',
    'InstanceType': 'ml.c4.xlarge',
    'VolumeSizeInGB': 400,
    'RuleParameters': {
        'source_s3_uri': 's3://bucket/custom_rules.py',
        'rule_to_invoke': 'ImproperActivation',
        'collection_names': 'relu_activations'
    }
}
]

```

To find a complete list of available Docker images for using the Debugger rules, see [Use Debugger Docker Images for Built-in or Custom Rules \(p. 1738\)](#). To find the key-value pairs for `RuleParameters`, see [List of Debugger Built-in Rules \(p. 1626\)](#).

## List of Debugger Built-in Rules

Use the Debugger built-in rules provided by Amazon SageMaker Debugger and analyze tensors emitted while training your models. The Debugger built-in rules monitor various common conditions that are critical for the success of a training job. You can call the built-in rules using [Amazon SageMaker Python SDK](#) or the low-level SageMaker API operations. Depending on deep learning frameworks of your choice, there are four scopes of validity for the built-in rules as shown in the following table.

### Note

The maximum numbers of built-in rules for a training job are 20 for `ProfilerRule` and 20 for `Rule`. SageMaker Debugger fully manages the built-in rules and analyzes your training job in parallel. For more information about billing, see the [Amazon SageMaker Studio is available at no additional charge](#) section of the [Amazon SageMaker Pricing](#) page.

### Important

To use the new Debugger features, you need to upgrade the SageMaker Python SDK and the SMDebug client library. In your iPython kernel, Jupyter notebook, or JupyterLab environment, run the following code to install the latest versions of the libraries and restart the kernel.

```
import sys
import IPython
!{sys.executable} -m pip install -U sagemaker smdebug
IPython.Application.instance().kernel.do_shutdown(True)
```

## Debugger ProfilerRule

The following rules are the Debugger built-in rules that are callable using the `ProfilerRule.sagemaker` classmethod.

### Debugger Built-in Rules for Generating Profiling Reports

Scope of Validity	Built-in Rules
Profiling Report for any SageMaker training job	<ul style="list-style-type: none"><li><a href="#">ProfilerReport (p. 1628)</a></li></ul>

### Debugger Built-in Rules for Monitoring Hardware System Resource Utilization (System Metrics)

Scope of Validity	Built-in Rules
Generic system monitoring rules for any SageMaker training job	<ul style="list-style-type: none"><li><a href="#">BatchSize (p. 1629)</a></li><li><a href="#">CPUBottleneck (p. 1631)</a></li><li><a href="#">GPUMemoryIncrease (p. 1632)</a></li><li><a href="#">IOBottleneck (p. 1633)</a></li><li><a href="#">LoadBalancing (p. 1634)</a></li><li><a href="#">LowGPUUtilization (p. 1635)</a></li><li><a href="#">OverallSystemUsage (p. 1636)</a></li></ul>

### Debugger Built-in Rules for Profiling Framework Metrics

Scope of Validity	Built-in Rules
Profiling rules for deep learning frameworks (TensorFlow and PyTorch)	<ul style="list-style-type: none"><li><a href="#">MaxInitializationTime (p. 1636)</a></li><li><a href="#">OverallFrameworkMetrics (p. 1637)</a></li></ul>

Scope of Validity	Built-in Rules
	<ul style="list-style-type: none"> <li>• <a href="#">StepOutlier (p. 1637)</a></li> </ul>

## Debugger Rule

The following rules are the Debugger built-in rules that are callable using the `Rule.sagemaker` classmethod.

### Debugger Built-in Rules for Generating Training Reports

Scope of Validity	Built-in Rules
Training Report for SageMaker XGboost training job	<ul style="list-style-type: none"> <li>• <a href="#">create_xgboost_report (p. 1638)</a></li> </ul>

### Debugger Built-in Rules for Debugging Model Training Data (Output Tensors)

Scope of Validity	Built-in Rules
Deep learning frameworks (TensorFlow, MXNet, and PyTorch)	<ul style="list-style-type: none"> <li>• <a href="#">dead_relu (p. 1639)</a></li> <li>• <a href="#">exploding_tensor (p. 1640)</a></li> <li>• <a href="#">poor_weight_initialization (p. 1641)</a></li> <li>• <a href="#">saturated_activation (p. 1643)</a></li> <li>• <a href="#">vanishing_gradient (p. 1646)</a></li> <li>• <a href="#">weight_update_ratio (p. 1647)</a></li> </ul>
Deep learning frameworks (TensorFlow, MXNet, and PyTorch) and the XGBoost algorithm	<ul style="list-style-type: none"> <li>• <a href="#">all_zero (p. 1648)</a></li> <li>• <a href="#">class_imbalance (p. 1650)</a></li> <li>• <a href="#">loss_not_decreasing (p. 1652)</a></li> <li>• <a href="#">overfit (p. 1654)</a></li> <li>• <a href="#">overtraining (p. 1655)</a></li> <li>• <a href="#">similar_across_runs (p. 1657)</a></li> <li>• <a href="#">stalled_training_rule (p. 1658)</a></li> <li>• <a href="#">tensor_variance (p. 1659)</a></li> <li>• <a href="#">unchanged_tensor (p. 1661)</a></li> </ul>
Deep learning applications	<ul style="list-style-type: none"> <li>• <a href="#">check_input_images (p. 1663)</a></li> <li>• <a href="#">nlp_sequence_ratio (p. 1664)</a></li> </ul>
XGBoost algorithm	<ul style="list-style-type: none"> <li>• <a href="#">confusion (p. 1666)</a></li> <li>• <a href="#">feature_importance_overweight (p. 1668)</a></li> <li>• <a href="#">tree_depth (p. 1669)</a></li> </ul>

**To use the built-in rules with default parameter values** – use the following configuration format:

```
from sagemaker.debugger import Rule, ProfilerRule, rule_configs

rules = [
    ProfilerRule.sagemaker(rule_configs.BuiltInRuleName_1()),
    ProfilerRule.sagemaker(rule_configs.BuiltInRuleName_2()),
    ...
]
```

```
    ProfilerRule.sagemaker(rule_configs.BuiltInRuleName_n())),
    Rule.sagemaker(rule_configs.built_in_rule_name_1()),
    Rule.sagemaker(rule_configs.built_in_rule_name_2()),
    ...
    Rule.sagemaker(rule_configs.built_in_rule_name_n()))
]
```

**To use the built-in rules with customizing the parameter values** – use the following configuration format:

```
from sagemaker.debugger import Rule, ProfilerRule, rule_configs

rules = [
    ProfilerRule.sagemaker(
        base_config=rule_configs.BuiltInRuleName(),
        rule_parameters={
            "key": "value"
        }
    )
    Rule.sagemaker(
        base_config=rule_configs.built_in_rule_name(),
        rule_parameters={
            "key": "value"
        }
    )
    collections_to_save=[
        CollectionConfig(
            name="tensor_collection_name",
            parameters={
                "key": "value"
            }
        )
    ]
]
```

To find available keys for the `rule_parameters` parameter, see the parameter description tables.

Sample rule configuration codes are provided for each built-in rule below the parameter description tables.

- For a full instruction and examples of using the Debugger built-in rules, see [Debugger Built-in Rules Example Code \(p. 1612\)](#).
- For a full instruction of using the built-in rules with the low-level SageMaker API operations, see [Configure Debugger Using Amazon SageMaker API \(p. 1615\)](#).

## ProfilerReport

The ProfilerReport rule invokes all of the built-in rules for monitoring and profiling. It creates a profiling report and updates when the individual rules are triggered. You can download a comprehensive profiling report while a training job is running or after the training job is complete. You can adjust the rule parameter values to customize sensitivity of the built-in monitoring and profiling rules. The following example code shows the basic format to adjust the built-in rule parameters through the ProfilerReport rule.

```
rules=[
    ProfilerRule.sagemaker(
        rule_configs.ProfilerReport(
            <BuiltInRuleName>_<parameter_name> = value
        )
    )
]
```

]

If you trigger this ProfilerReport rule without any customized parameter as shown in the following example code, then the ProfilerReport rule triggers all of the built-in rules for monitoring and profiling with their default parameter values.

```
rules=[ProfilerRule.sagemaker(rule_configs.ProfilerReport())]
```

The following example code shows how to specify and adjust the CPUBottleneck rule's `cpu_threshold` parameter and the IOBottleneck rule's `threshold` parameter.

```
rules=[
    ProfilerRule.sagemaker(
        rule_configs.ProfilerReport(
            CPUBottleneck_cpu_threshold = 90,
            IOBottleneck_threshold = 90
        )
    )
]
```

### Parameter Descriptions for the OverallSystemUsage Rule

Parameter Name	Description
<code>base_trial</code>	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
<code>&lt;BuiltInRuleName&gt;_&lt;parameter_name&gt;</code>	<p>Customizable parameter to adjust thresholds of other built-in monitoring and profiling rules.</p> <p><b>Optional</b></p> <p>Default value: None</p>

## BatchSize

The BatchSize rule helps detect if GPU is underutilized due to a small batch size. To detect this issue, this rule monitors the average CPU utilization, GPU utilization, and GPU memory utilization. If utilization on CPU, GPU, and GPU memory is low on average, it may indicate that the training job can either run on a smaller instance type or can run with a bigger batch size. This analysis does not work for frameworks that heavily overallocate memory. However, increasing the batch size can lead to processing or data loading bottlenecks because more data preprocessing time is required in each iteration.

### Parameter Descriptions for the BatchSize Rule

Parameter Name	Description
<code>base_trial</code>	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p>

<b>Parameter Name</b>	<b>Description</b>
	Valid values: String
cpu_threshold_p95	<p>Defines the threshold for 95th quantile of CPU utilization in percentage.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 70 (in percentage)</p>
gpu_threshold_p95	<p>Defines the threshold for 95th quantile of GPU utilization in percentage.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 70 (in percentage)</p>
gpu_memory_threshold_p95	<p>Defines the threshold for 95th quantile of GPU memory utilization in percentage.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 70 (in percentage)</p>
patience	<p>Defines the number of data points to skip until the rule starts evaluation. The first several steps of training jobs usually show high volume of data processes, so keep the rule patient and prevent it from being invoked too soon with a given number of profiling data that you specify with this parameter.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 100</p>
window	<p>Window size for computing quantiles.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 500</p>
scan_interval_us	<p>Time interval that timeline files are scanned.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 6000000 (in microseconds)</p>

## CPUBottleneck

The CPUBottleneck rule helps detect if GPU is underutilized due to CPU bottlenecks. Rule returns True if number of CPU bottlenecks exceeds a predefined threshold.

### Parameter Descriptions for the CPUBottleneck Rule

Parameter Name	Description
<code>base_trial</code>	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
<code>threshold</code>	<p>Defines the threshold for proportion of bottlenecked time to the total training time. If the proportion exceeds the percentage specified to the threshold parameter, the rule switches the rule status to True.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 50 (in percentage)</p>
<code>gpu_threshold</code>	<p>A threshold that defines low GPU utilization.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 10 (in percentage)</p>
<code>cpu_threshold</code>	<p>A threshold that defines high CPU utilization.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 90 (in percentage)</p>
<code>patience</code>	<p>Defines the number of data points to skip until the rule starts evaluation. The first several steps of training jobs usually show high volume of data processes, so keep the rule patient and prevent it from being invoked too soon with a given number of profiling data that you specify with this parameter.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 100</p>
<code>scan_interval_us</code>	Time interval with which timeline files are scanned.

Parameter Name	Description
	<b>Optional</b>  Valid values: Integer  Default values: 60000000 (in microseconds)

## GPUMemoryIncrease

The GPUMemoryIncrease rule helps detect a large increase in memory usage on GPUs.

### Parameter Descriptions for the GPUMemoryIncrease Rule

Parameter Name	Description
base_trial	The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.  <b>Required</b>  Valid values: String  Default value: N/A
increase	Defines the threshold for absolute memory increase.  <b>Optional</b>  Valid values: Integer  Default value: 10 (in percentage)
patience	Defines the number of data points to skip until the rule starts evaluation. The first several steps of training jobs usually show high volume of data processes, so keep the rule patient and prevent it from being invoked too soon with a given number of profiling data that you specify with this parameter.  <b>Optional</b>  Valid values: Integer  Default values: 100
window	Window size for computing quantiles.  <b>Optional</b>  Valid values: Integer  Default values: 500
scan_interval_us	Time interval that timeline files are scanned.  <b>Optional</b>  Valid values: Integer

Parameter Name	Description
	Default values: 60000000 (in microseconds)

## IOBottleneck

This rule helps to detect if GPU is underutilized due to data IO bottlenecks. Rule returns True if number of IO bottlenecks exceeds a predefined threshold.

### Parameter Descriptions for the IOBottleneck Rule

Parameter Name	Description
base_trial	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
threshold	<p>Defines the threshold when Rule to return True.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 50 (in percentage)</p>
gpu_threshold	<p>A threshold that defines when GPU is considered underutilized.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 70 (in percentage)</p>
io_threshold	<p>A threshold that defines high IO wait time.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 50 (in percentage)</p>
patience	<p>Defines the number of data points to skip until the rule starts evaluation. The first several steps of training jobs usually show high volume of data processes, so keep the rule patient and prevent it from being invoked too soon with a given number of profiling data that you specify with this parameter.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 1000</p>

Parameter Name	Description
scan_interval_us	<p>Time interval that timeline files are scanned.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 60000000 (in microseconds)</p>

## LoadBalancing

The LoadBalancing rule helps detect issues in workload balancing among multiple GPUs.

### Parameter Descriptions for the LoadBalancing Rule

Parameter Name	Description
base_trial	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
threshold	<p>Defines the workload percentage.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 0 . 5 (unitless proportion)</p>
patience	<p>Defines the number of data points to skip until the rule starts evaluation. The first several steps of training jobs usually show high volume of data processes, so keep the rule patient and prevent it from being invoked too soon with a given number of profiling data that you specify with this parameter.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 10</p>
scan_interval_us	<p>Time interval that timeline files are scanned.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 60000000 (in microseconds)</p>

## LowGPUUtilization

The LowGPUUtilization rule helps detect if GPU utilization is low or suffers from fluctuations. This is checked for each GPU on each worker. Rule returns True if 95th quantile is below threshold\_p95 which indicates underutilization. Rule returns true if 95th quantile is above threshold\_p95 and 5th quantile is below threshold\_p5 which indicates fluctuations.

### Parameter Descriptions for the LowGPUUtilization Rule

Parameter Name	Description
<code>base_trial</code>	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
<code>threshold_p95</code>	<p>A threshold for 95th quantile below which GPU is considered to be underutilized.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 70 (in percentage)</p>
<code>threshold_p5</code>	<p>A threshold for 5th quantile. Default is 10 percent.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 10 (in percentage)</p>
<code>patience</code>	<p>Defines the number of data points to skip until the rule starts evaluation. The first several steps of training jobs usually show high volume of data processes, so keep the rule patient and prevent it from being invoked too soon with a given number of profiling data that you specify with this parameter.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 1000</p>
<code>window</code>	<p>Window size for computing quantiles.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 500</p>
<code>scan_interval_us</code>	<p>Time interval that timeline files are scanned.</p> <p><b>Optional</b></p>

Parameter Name	Description
	Valid values: Integer
	Default values: 60000000 (in microseconds)

## OverallSystemUsage

The OverallSystemUsage rule measures overall system usage per worker node. The rule currently only aggregates values per node and computes their percentiles.

### Parameter Descriptions for the OverallSystemUsage Rule

Parameter Name	Description
<code>base_trial</code>	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
<code>scan_interval_us</code>	<p>Time interval to scan timeline files.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 60000000 (in microseconds)</p>

## MaxInitializationTime

The MaxInitializationTime rule helps detect if the training initialization is taking too much time. The rule waits until the first step is available.

### Parameter Descriptions for the MaxInitializationTime Rule

Parameter Name	Description
<code>base_trial</code>	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
<code>threshold</code>	<p>Defines the threshold in minutes to wait for the first step to become available.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 20 (in minutes)</p>

Parameter Name	Description
scan_interval_us	<p>Time interval with which timeline files are scanned.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 60000000 (in microseconds)</p>

## OverallFrameworkMetrics

The OverallFrameworkMetrics rule summarizes the time spent on framework metrics, such as forward and backward pass, and data loading.

### Parameter Descriptions for the OverallFrameworkMetrics Rule

Parameter Name	Description
base_trial	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
scan_interval_us	<p>Time interval to scan timeline files.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default values: 60000000 (in microseconds)</p>

## StepOutlier

The StepOutlier rule helps detect outliers in step durations. This rule returns `True` if there are outliers with step durations larger than `stddev` sigmas of the entire step durations in a time range.

### Parameter Descriptions for the StepOutlier Rule

Parameter Name	Description
base_trial	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
stddev	<p>Defines a factor by which to multiply the standard deviation. For example, the rule is invoked by default when a step duration is larger or smaller than 5 times the standard deviation.</p>

Parameter Name	Description
	<b>Optional</b> Valid values: Integer Default value: 5 (in minutes)
mode	Mode under which steps have been saved and on which Rule should run on. Per default rule will run on steps from EVAL and TRAIN phase <b>Optional</b> Valid values: Integer Default value: 5 (in minutes)
n_outliers	How many outliers to ignore before rule returns True <b>Optional</b> Valid values: Integer Default value: 10
scan_interval_us	Time interval with which timeline files are scanned. <b>Optional</b> Valid values: Integer Default values: 60000000 (in microseconds)

## CreateXgboostReport

The CreateXgboostReport rule collects output tensors from an XGBoost training job and autogenerated a comprehensive training report. You can download a comprehensive profiling report while a training job is running or after the training job is complete, and check progress of training or the final result of the training job. The CreateXgboostReport rule collects the following output tensors by default:

- `hyperparameters` – Saves at the first step
- `metrics` – Saves loss and accuracy every 5 steps
- `feature_importance` – Saves every 5 steps
- `predictions` – Saves every 5 steps
- `labels` – Saves every 5 steps

### Parameter Descriptions for the CreateXgboostReport Rule

Parameter Name	Description
<code>base_trial</code>	The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.

Parameter Name	Description
	<b>Required</b>  Valid values: String

```

rules=[
    Rule.sagemaker(
        rule_configs.create_xgboost_report()
    )
]

```

## DeadRelu

This rule detects when the percentage of rectified linear unit (ReLU) activation functions in a trial are considered dead because their activation activity has dropped below a threshold. If the percent of inactive ReLUs in a layer is greater than the `threshold_layer` value of inactive ReLUs, the rule returns `True`.

### Parameter Descriptions for the DeadRelu Rule

Parameter Name	Description
<code>base_trial</code>	The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.  <b>Required</b>  Valid values: String
<code>tensor_regex</code>	A list of regex patterns used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.  <b>Optional</b>  Valid values: List of strings or a comma-separated string  Default value: ".*relu_output"
<code>threshold_inactivity</code>	Defines a level of activity below which a ReLU is considered to be dead. A ReLU might be active in the beginning of a trial and then slowly die during the training process. If the ReLU is active less than the <code>threshold_inactivity</code> , it is considered to be dead.  <b>Optional</b>  Valid values: Float  Default values: 1.0 (in percentage)

Parameter Name	Description
threshold_layer	<p>Returns <code>True</code> if the percentage of inactive ReLUs in a layer is greater than <code>threshold_layer</code>.</p> <p>Returns <code>False</code> if the percentage of inactive ReLUs in a layer is less than <code>threshold_layer</code>.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default values: 50.0 (in percentage)</p>

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.dead_relu(),
        rule_parameters={
            "tensor_regex": ".*relu_output|.*ReLU_output",
            "threshold_inactivity": "1.0",
            "threshold_layer": "50.0"
        }
    ),
    collections_to_save=[
        CollectionConfig(
            name="custom_relu_collection",
            parameters={
                "include_regex": ".*relu_output|.*ReLU_output",
                "save_interval": "500"
            }
        )
    ]
]

```

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

**Note**

This rule is not available for the XGBoost algorithm.

## ExplodingTensor

This rule detects whether the tensors emitted during training have non-finite values, either infinite or NaN (not a number). If a non-finite value is detected, the rule returns `True`.

### Parameter Descriptions for the ExplodingTensor Rule

Parameter Name	Description
base_trial	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
collection_names	The list of collection names whose tensors the rule inspects.

Parameter Name	Description
	<p><b>Optional</b></p> <p>Valid values: String</p> <p>Default value: None</p>
<code>tensor_regex</code>	<p>A list of regex patterns used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</p>
	<p><b>Optional</b></p> <p>Valid values: String</p> <p>Default value: None</p>
<code>only_nan</code>	<p>True to monitor the <code>base_trial</code> tensors only for NaN values and not for infinity.</p> <p>False to treat both NaN and infinity as exploding values and to monitor for both.</p>
	<p><b>Optional</b></p> <p>Default value: False</p>

```
built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.exploding_tensor(),
        rule_parameters={
            "tensor_regex": ".*\ngradient",
            "only_nan": "False"
        }
    ),
    collections_to_save=[
        CollectionConfig(
            name="gradients",
            parameters={
                "save_interval": "500"
            }
        )
    ]
]
```

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

## Note

This rule is not available for the XGBoost algorithm.

# PoorWeightInitialization

This rule detects if your model parameters have been poorly initialized.

Good initialization breaks the symmetry of the weights and gradients in a neural network and maintains commensurate activation variances across layers. Otherwise, the neural network doesn't learn effectively. Initializers like Xavier aim to keep variance constant across activations, which is especially relevant for training very deep neural nets. Too small an initialization can lead to vanishing gradients. Too large an initialization can lead to exploding gradients. This rule checks the variance of activation inputs across layers, the distribution of gradients, and the loss convergence for the initial steps to determine if a neural network has been poorly initialized.

### Parameter Descriptions for the PoorWeightInitialization Rule

Parameter Name	Description
<code>base_trial</code>	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
<code>activation_inputs_regex</code>	<p>A list of regex patterns used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</p> <p><b>Optional</b></p> <p>Valid values: String</p> <p>Default value: ". *relu_input"</p>
<code>threshold</code>	<p>If the ratio between minimum and maximum variance of weights per layer exceeds the threshold at a step, the rule returns <code>True</code>.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default value: 10 . 0</p>
<code>distribution_range</code>	<p>If the minimum difference between 5th and 95th percentiles of the gradient distribution is less than the <code>distribution_range</code>, the rule returns <code>True</code>.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default value: 0 . 001</p>
<code>patience</code>	<p>The number of steps to wait until the loss is considered to be no longer decreasing.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p>

Parameter Name	Description
	Default value: 5
steps	<p>The number of steps this rule analyzes. You typically need to check only the first few iterations.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default value: 10</p>

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.poor_weight_initialization(),
        rule_parameters={
            "activation_inputs_regex": ".*relu_input|.*ReLU_input",
            "threshold": "10.0",
            "distribution_range": "0.001",
            "patience": "5",
            "steps": "10"
        },
        collections_to_save=[
            CollectionConfig(
                name="custom_relu_collection",
                parameters={
                    "include_regex": ".*relu_input|.*ReLU_input",
                    "save_interval": "500"
                }
            )
        ]
    )
]

```

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

**Note**

This rule is not available for the XGBoost algorithm.

## SaturatedActivation

This rule detects if the tanh and sigmoid activation layers are becoming saturated. An activation layer is saturated when the input of the layer is close to the maximum or minimum of the activation function. The minimum and maximum of the tanh and sigmoid activation functions are defined by their respective `min_threshold` and `max_threshold` values. If the activity of a node drops below the `threshold_inactivity` percentage, it is considered saturated. If more than a `threshold_layer` percent of the nodes are saturated, the rule returns `True`.

### Parameter Descriptions for the SaturatedActivation Rule

Parameter Name	Description
base_trial	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p>

Parameter Name	Description
	Valid values: String
<code>collection_names</code>	<p>The list of collection names whose tensors the rule inspects.</p> <p><b>Optional</b></p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>tensor_regex</code>	<p>A list of regex patterns used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</p> <p><b>Optional</b></p> <p>Valid values: String</p> <p>Default value: ".*tanh_input .*sigmoid_input".</p>
<code>threshold_tanh_min</code>	<p>The minimum and maximum thresholds that define the extremes of the input for a tanh activation function, defined as: (<code>min_threshold</code>, <code>max_threshold</code>). The default values are determined based on a vanishing gradient threshold of 0.0000001.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default values: -9 . 4999</p>
<code>threshold_tanh_max</code>	<p>The minimum and maximum thresholds that define the extremes of the input for a tanh activation function, defined as: (<code>min_threshold</code>, <code>max_threshold</code>). The default values are determined based on a vanishing gradient threshold of 0.0000001.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default values: 9 . 4999</p>

Parameter Name	Description
threshold_sigmoid_min	<p>The minimum and maximum thresholds that define the extremes of the input for a sigmoid activation function, defined as: (<code>min_threshold</code>, <code>max_threshold</code>). The default values are determined based on a vanishing gradient threshold of 0.0000001.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default values: -23</p>
threshold_sigmoid_max	<p>The minimum and maximum thresholds that define the extremes of the input for a sigmoid activation function, defined as: (<code>min_threshold</code>, <code>max_threshold</code>). The default values are determined based on a vanishing gradient threshold of 0.0000001.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default values: 16 . 99999</p>
threshold_inactivity	<p>The percentage of inactivity below which the activation layer is considered to be saturated. The activation might be active in the beginning of a trial and then slowly become less active during the training process.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default values: 1 . 0</p>
threshold_layer	<p>Returns <code>True</code> if the number of saturated activations in a layer is greater than the <code>threshold_layer</code> percentage.</p> <p>Returns <code>False</code> if the number of saturated activations in a layer is less than the <code>threshold_layer</code> percentage.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default values: 50 . 0</p>

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.saturated_activation(),
        rule_parameters={
            "tensor_regex": ".*tanh_input|.*sigmoid_input",

```

```

        "threshold_tanh_min": "-9.4999",
        "threshold_tanh_max": "9.4999",
        "threshold_sigmoid_min": "-23",
        "threshold_sigmoid_max": "16.99999",
        "threshold_inactivity": "1.0",
        "threshold_layer": "50.0"
    },
    collections_to_save=[
        CollectionConfig(
            name="custom_activations_collection",
            parameters={
                "include_regex": ".*tanh_input/.*sigmoid_input"
                "save_interval": "500"
            }
        )
    ]
]

```

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

**Note**

This rule is not available for the XGBoost algorithm.

## VanishingGradient

This rule detects if the gradients in a trial become extremely small or drop to a zero magnitude. If the mean of the absolute values of the gradients drops below a specified threshold, the rule returns `True`.

### Parameters Descriptions for the VanishingGradient Rule

Parameter Name	Description
<code>base_trial</code>	The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.  <b>Required</b>  Valid values: String
<code>threshold</code>	The value at which the gradient is determined to be vanishing.  <b>Optional</b>  Valid values: Float  Default value: 0.0000001.

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.vanishing_gradient(),
        rule_parameters={
            "threshold": "0.0000001"
        },
        collections_to_save=[
            CollectionConfig(
                name="gradients",

```

```

        parameters={
            "save_interval": "500"
        }
    ]
}
]
```

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

**Note**

This rule is not available for the XGBoost algorithm.

## WeightUpdateRatio

This rule keeps track of the ratio of updates to weights during training and detects if that ratio gets too large or too small. If the ratio of updates to weights is larger than the `large_threshold` value or if this ratio is smaller than `small_threshold`, the rule returns `True`.

Conditions for training are best when the updates are commensurate to gradients. Excessively large updates can push the weights away from optimal values, and very small updates result in very slow convergence. This rule requires weights to be available for two training steps, and `train.save_interval` needs to be set equal to `num_steps`.

### Parameter Descriptions for the WeightUpdateRatio Rule

Parameter Name,	Description
<code>base_trial</code>	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
<code>num_steps</code>	<p>The number of steps across which the rule checks to determine if the tensor has changed.</p> <p>The number of steps across which you want to compare the weight ratios. If you pass no value, the rule runs by default against the current step and the immediately previous saved step. If you override the default by passing a value for this parameter, the comparison is done between weights at step <code>s</code> and at a step <math>\geq s - num\_steps</math>.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: None</p>
<code>large_threshold</code>	<p>The maximum value that the ratio of updates to weight can take before the rule returns <code>True</code>.</p> <p><b>Optional</b></p>

<b>Parameter Name,</b>	<b>Description</b>
	Valid values: Float  Default value: 10 . 0
small_threshold	The minimum value that the ratio of updates to weight can take, below which the rule returns True.  <b>Optional</b>  Valid values: Float  Default value: 0 . 00000001
epsilon	A small constant used to ensure that Debugger does not divide by zero when computing the ratio updates to weigh.  <b>Optional</b>  Valid values: Float  Default value: 0 . 000000001

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.weight_update_ratio(),
        rule_parameters={
            "num_steps": "100",
            "large_threshold": "10.0",
            "small_threshold": "0.00000001",
            "epsilon": "0.000000001"
        },
        collections_to_save=[
            CollectionConfig(
                name="weights",
                parameters={
                    "train.save_interval": "100"
                }
            )
        ]
    )
]

```

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

**Note**

This rule is not available for the XGBoost algorithm.

## AllZero

This rule detects if all or a specified percentage of the tensor values are zero.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm. You must specify either the `collection_names` or `tensor_regex` parameter. If both the parameters are specified, the rule inspects the union of tensors from both sets.

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

### Parameters Descriptions for the AllZero Rule

Parameter Name	Description
<code>base_trial</code>	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
<code>collection_names</code>	<p>The list of collection names whose tensors the rule inspects.</p> <p><b>Optional</b></p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>tensor_regex</code>	<p>A list of regex patterns used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</p> <p><b>Optional</b></p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>threshold</code>	<p>Specifies the percentage of values in the tensor that needs to be zero for this rule to be invoked.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default value: 100 (in percentage)</p>

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.all_zero(),
        rule_parameters={
            "tensor_regex": ".*",
            "threshold": "100"
        },
        collections_to_save=[
            CollectionConfig(
                name="all",
                parameters={


```

```

        "save_interval": "500"
    }
}
]
```

## ClassImbalance

This rule measures sampling imbalances between classes and throws errors if the imbalance exceeds a threshold or if too many mispredictions for underrepresented classes occur as a result of the imbalance.

Classification models require well-balanced classes in the training dataset or a proper weighting/sampling of classes during training. The rule performs the following checks:

- It counts the occurrences per class. If the ratio of number of samples between smallest and largest class is larger than the `threshold_imbalance`, an error is thrown.
- It checks the prediction accuracy per class. If resampling or weighting has not been correctly applied, then the model can reach high accuracy for the class with many training samples, but low accuracy for the classes with few training samples. If a fraction of mispredictions for a certain class is above `threshold_misprediction`, an error is thrown.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

### Parameter Descriptions for the ClassImbalance Rule

Parameter Name	Description
<code>base_trial</code>	The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.  <b>Required</b>  Valid values: String
<code>threshold_imbalance</code>	The acceptable imbalance between the number of samples in the smallest class and in the largest class. Exceeding this threshold value throws an error.  <b>Optional</b>  Valid values: Float  Default value: 10
<code>threshold_misprediction</code>	A limit on the fraction of mispredictions allowed for each class. Exceeding this threshold throws an error. The underrepresented classes are most at risk of crossing this threshold.  <b>Optional</b>  Valid values: Float

Parameter Name	Description
	Default value: 0 . 7
<code>samples</code>	The number of labels that have to be processed before an imbalance is evaluated. The rule might not be triggered until it has seen sufficient samples across several steps. The more classes that your dataset contains, the larger this sample number should be.
	<b>Optional</b>
	Valid values: Integer
	Default value: 500 (assuming a dataset like MNIST with 10 classes)
<code>argmax</code>	If <code>True</code> , <code>np.argmax</code> is applied to the prediction tensor. Required when you have a vector of probabilities for each class. It is used to determine which class has the highest probability.
	<b>Conditional</b>
	Valid values: Boolean
	Default value: <code>False</code>
<code>labels_regex</code>	The name of the tensor that contains the labels.
	<b>Optional</b>
	Valid values: String
	Default value: ". *labels"
<code>predictions_regex</code>	The name of the tensor that contains the predictions.
	<b>Optional</b>
	Valid values: String
	Default value: ". *predictions"

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.class_imbalance(),
        rule_parameters={
            "threshold_imbalance": "10",
            "threshold_misprediction": "0.7",
            "samples": "500",
            "argmax": "False",
            "labels_regex": ". *labels",
            "predictions_regex": ". *predictions"
        },
        collections_to_save=[
            CollectionConfig(
                name="custom_output_collection",

```

```

        parameters={
            "include_regex": ".*labels/.*predictions",
            "save_interval": "500"
        }
    ]
]
]
```

## LossNotDecreasing

This rule detects when the loss is not decreasing in value at an adequate rate. These losses must be scalars.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm. You must specify either the `collection_names` or `tensor_regex` parameter. If both the parameters are specified, the rule inspects the union of tensors from both sets.

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

### Parameter Descriptions for the LossNotDecreasing Rule

Parameter Name	Description
<code>base_trial</code>	The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.  <b>Required</b>  Valid values: String
<code>collection_names</code>	The list of collection names whose tensors the rule inspects.  <b>Optional</b>  Valid values: List of strings or a comma-separated string  Default value: None
<code>tensor_regex</code>	A list of regex patterns that is used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.  <b>Optional</b>  Valid values: List of strings or a comma-separated string  Default value: None
<code>use_losses_collection</code>	If set to <code>True</code> , looks for losses in the collection named "losses" when the collection is present.

Parameter Name	Description
	<b>Optional</b> Valid values: Boolean Default value: True
<code>num_steps</code>	The minimum number of steps after which the rule checks if the loss has decreased. Rule evaluation happens every <code>num_steps</code> . The rule compares the loss for this step with the loss at a step which is at least <code>num_steps</code> behind the current step. For example, suppose that the loss is being saved every three steps, but <code>num_steps</code> is set to 10. At step 21, loss for step 21 is compared with loss for step 9. The next step at which loss is checked is step 33, because ten steps after step 21 is step 31, and at step 31 and step 32 loss is not saved. <b>Optional</b> Valid values: Integer Default value: 10
<code>diff_percent</code>	The minimum percentage difference by which the loss should decrease between <code>num_steps</code> . <b>Optional</b> Valid values: <code>0 . 0 &lt; float &lt; 100</code> Default value: <code>0 . 1</code> (in percentage)
<code>increase_threshold_percent</code>	The maximum threshold percent that loss is allowed to increase in case loss has been increasing <b>Optional</b> Valid values: <code>0 &lt; float &lt; 100</code> Default value: <code>5</code> (in percentage)
<code>mode</code>	The name of the Debugger mode to query tensor values for rule checking. If this is not passed, the rule checks in order by default for the mode <code>EVAL</code> , then <code>mode . TRAIN</code> , and then <code>mode . GLOBAL</code> . <b>Optional</b> Valid values: String ( <code>EVAL</code> , <code>TRAIN</code> , or <code>GLOBAL</code> ) Default value: <code>GLOBAL</code>

```
built_in_rules = [
```

```

Rule.sagemaker(
    base_config=rule_configs.loss_not_decreasing(),
    rule_parameters={
        "tensor_regex": ".*",
        "use_losses_collection": "True",
        "num_steps": "10",
        "diff_percent": "0.1",
        "increase_threshold_percent": "5",
        "mode": "GLOBAL"
    },
    collections_to_save=[
        CollectionConfig(
            name="losses",
            parameters={
                "save_interval": "500"
            }
        )
    ]
)
]

```

## Overfit

This rule detects if your model is being overfit to the training data by comparing the validation and training losses.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

**Note**

A standard way to prevent overfitting is to regularize your model.

### Parameter Descriptions for the Overfit Rule

Parameter Name	Description
<code>base_trial</code>	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
<code>tensor_regex</code>	<p>A list of regex patterns used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</p> <p><b>Optional</b></p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>

Parameter Name	Description
<code>start_step</code>	<p>The step from which to start comparing the validation and training loss.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 0</p>
<code>patience</code>	<p>The number of steps for which the <code>ratio_threshold</code> is allowed to exceed the value set before the model is considered to be overfit.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 1</p>
<code>ratio_threshold</code>	<p>The maximum ratio of the difference between the mean validation loss and mean training loss to the mean training loss. If this threshold is exceeded for a patience number of steps, the model is being overfit and the rule returns <code>True</code>.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default value: 0.1</p>

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.overfit(),
        rule_parameters={
            "tensor_regex": ".*",
            "start_step": "0",
            "patience": "1",
            "ratio_threshold": "0.1"
        },
        collections_to_save=[
            CollectionConfig(
                name="losses",
                parameters={
                    "train.save_interval": "100",
                    "eval.save_interval": "10"
                }
            )
        ]
    )
]

```

## Overtraining

This rule detects if a model is being overtrained. After a number of training iterations on a well-behaved model (both training and validation loss decrease), the model approaches to a minimum of the loss function and does not improve anymore. If the model continues training it can happen that validation

loss starts increasing, because the model starts overfitting. This rule sets up thresholds and conditions to determine if the model is not improving, and prevents overfitting problems due to overtraining.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

**Note**

Overtraining can be avoided by early stopping. For information on early stopping, see [Stop Training Jobs Early \(p. 1764\)](#). For an example that shows how to use spot training with Debugger, see [Enable Spot Training with Amazon SageMaker Debugger](#).

### Parameter Descriptions for the Overtraining Rule

Parameter Name	Description
base_trial	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
patience_train	<p>The number of steps to wait before the training loss is considered to not to be improving anymore.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 5</p>
patience_validation	<p>The number of steps to wait before the validation loss is considered to not to be improving anymore.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 10</p>
delta	<p>The minimum threshold by how much the error should improve before it is considered as a new optimum.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default value: 0 . 01</p>

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.overtraining(),
        rule_parameters={
            "patience_train": "5",
            "patience_validation": "10",

```

```

        "delta": "0.01"
    },
    collections_to_save:[
        CollectionConfig(
            name="losses",
            parameters={
                "save_interval": "500"
            }
        )
    ]
]

```

## SimilarAcrossRuns

This rule compares tensors gathered from a base trial with tensors from another trial.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

### Parameter Descriptions for the SimilarAcrossRuns Rule

Parameter Name	Description
base_trial	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
other_trial	<p>A completed training job name whose tensors you want to compare to those tensors gathered from the current <code>base_trial</code>.</p> <p><b>Required</b></p> <p>Valid values: String</p>
collection_names	<p>The list of collection names whose tensors the rule inspects.</p> <p><b>Optional</b></p> <p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
tensor_regex	<p>A list of regex patterns used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.</p>

Parameter Name	Description
	<b>Optional</b>  Valid values: List of strings or a comma-separated string  Default value: None

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.similar_across_runs(),
        rule_parameters={
            "other_trial": "<specify-another-job-name>",
            "collection_names": "losses",
            "tensor_regex": ".*"
        },
        collections_to_save=[
            CollectionConfig(
                name="losses",
                parameters={
                    "save_interval": "500"
                }
            )
        ]
    )
]

```

## StalledTrainingRule

StalledTrainingRule detects if there is no progress made on training job, and stops the training job if the rule fires. This rule requires tensors to be periodically saved in a time interval defined by its `threshold` parameter. This rule keeps on monitoring for new tensors, and if no new tensor has been emitted for threshold interval rule gets fired.

### Parameter Descriptions for the StalledTrainingRule Rule

Parameter Name	Description
<code>base_trial</code>	The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.  <b>Required</b>  Valid values: String
<code>threshold</code>	A threshold that defines by how much time in seconds the rule waits for a tensor output until it fires a stalled training issue. Default value is 1800 seconds.  <b>Optional</b>  Valid values: Integer  Default value: 1800
<code>stop_training_on_fire</code>	If set to <code>True</code> , watches if the base training job outputs tensors in " <code>threshold</code> " seconds.

Parameter Name	Description
	<b>Optional</b> Valid values: Boolean Default value: <code>False</code>
<code>training_job_name_prefix</code>	The prefix of base training job name. If <code>stop_training_on_fire</code> is true, the rule searches for SageMaker training jobs with this prefix in the same account. If there is an inactivity found, the rule takes a <code>StopTrainingJob</code> action. Note if there are multiple jobs found with same prefix, the rule skips termination. It is important that the prefix is set unique per each training job. <b>Optional</b> Valid values: String

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.stalled_training_rule(),
        rule_parameters={
            "threshold": "1800",
            "stop_training_on_fire": "True",
            "training_job_name_prefix": "<specify-training-base-job-name>"
        },
        collections_to_save=[
            CollectionConfig(
                name="losses",
                parameters={
                    "save_interval": "500"
                }
            )
        ]
    )
]

```

## TensorVariance

This rule detects if you have tensors with very high or low variances. Very high or low variances in a tensor could lead to neuron saturation, which reduces the learning ability of the neural network. Very high variance in tensors can also eventually lead to exploding tensors. Use this rule to detect such issues early.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm. You must specify either the `collection_names` or `tensor_regex` parameter. If both the parameters are specified, the rule inspects the union of tensors from both sets.

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

## Parameter Descriptions for the TensorVariance Rule

Parameter Name	Description
<code>base_trial</code>	The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.  <b>Required</b>  Valid values: String
<code>collection_names</code>	The list of collection names whose tensors the rule inspects.  <b>Optional</b>  Valid values: List of strings or a comma-separated string  Default value: None
<code>tensor_regex</code>	A list of regex patterns used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.  <b>Optional</b>  Valid values: List of strings or a comma-separated string  Default value: None
<code>max_threshold</code>	The threshold for the upper bound of tensor variance.  <b>Optional</b>  Valid values: Float  Default value: None
<code>min_threshold</code>	The threshold for the lower bound of tensor variance.  <b>Optional</b>  Valid values: Float  Default value: None

```
built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.tensor_variance(),
        rule_parameters={
            "collection_names": "weights",

```

```

        "max_threshold": "10",
        "min_threshold": "0.00001",
    },
    collections_to_save=[
        CollectionConfig(
            name="weights",
            parameters={
                "save_interval": "500"
            }
        )
    ]
]

```

## UnchangedTensor

This rule detects whether a tensor is no longer changing across steps.

This rule runs the [numpy.allclose](#) method to check if the tensor isn't changing.

This rule can be applied either to one of the supported deep learning frameworks (TensorFlow, MXNet, and PyTorch) or to the XGBoost algorithm. You must specify either the `collection_names` or `tensor_regex` parameter. If both the parameters are specified, the rule inspects the union of tensors from both sets.

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

### Parameter Descriptions for the UnchangedTensor Rule

Parameter Name	Description
<code>base_trial</code>	The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.  <b>Required</b>  Valid values: String
<code>collection_names</code>	The list of collection names whose tensors the rule inspects.  <b>Optional</b>  Valid values: List of strings or a comma-separated string  Default value: None
<code>tensor_regex</code>	A list of regex patterns used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.  <b>Optional</b>

Parameter Name	Description
	<p>Valid values: List of strings or a comma-separated string</p> <p>Default value: None</p>
<code>num_steps</code>	<p>The number of steps across which the rule checks to determine if the tensor has changed.</p> <p>This checks the last <code>num_steps</code> that are available. They don't need to be consecutive. If <code>num_steps</code> is 2, at step s it doesn't necessarily check for s-1 and s. If s-1 isn't available, it checks the last available step along with s. In that case, it checks the last available step with the current step.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 3</p>
<code>rtol</code>	<p>The relative tolerance parameter to be passed to the <code>numpy.allclose</code> method.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default value: <code>1e-05</code></p>
<code>atol</code>	<p>The absolute tolerance parameter to be passed to the <code>numpy.allclose</code> method.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default value: <code>1e-08</code></p>
<code>equal_nan</code>	<p>Whether to compare NaNs as equal. If <code>True</code>, NaNs in input array a are considered equal to NaNs in input array b in the output array. This parameter is passed to the <code>numpy.allclose</code> method.</p> <p><b>Optional</b></p> <p>Valid values: Boolean</p> <p>Default value: <code>False</code></p>

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.unchanged_tensor(),
        rule_parameters={
            "collection_names": "losses",
            "tensor_regex": "",
            "num_steps": "3",
            "rtol": "1e-05",
        }
    )
]

```

```

        "atol": "1e-08",
        "equal_nan": "False"
    },
    collections_to_save:[
        CollectionConfig(
            name="losses",
            parameters={
                "save_interval": "500"
            }
        )
    ]
]

```

## CheckInputImages

This rule checks if input images have been correctly normalized. Specifically, it detects if the mean of the sample data differs by more than a threshold value from zero. Many computer vision models require that input data has a zero mean and unit variance.

This rule is applicable to deep learning applications.

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

### Parameter Descriptions for the CheckInputImages Rule

Parameter Name	Description
base_trial	<p>The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.</p> <p><b>Required</b></p> <p>Valid values: String</p>
threshold_mean	<p>A threshold that defines by how much mean of the input data can differ from 0.</p> <p><b>Optional</b></p> <p>Valid values: Float</p> <p>Default value: 0 . 2</p>
threshold_samples	<p>The number of images that have to be sampled before an error can be thrown. If the value is too low, the estimation of the dataset mean will be inaccurate.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 500</p>
regex	<p>The name of the input data tensor.</p> <p><b>Optional</b></p> <p>Valid values: String</p>

Parameter Name	Description
	<p>Default value: ".*hybridsequential0_input_0" (the name of the input tensor for Apache MXNet models using HybridSequential)</p>
channel	<p>The position of the color channel in the input tensor shape array.</p> <p><b>Optional</b></p> <p>Valid values: Integer</p> <p>Default value: 1 (for example, MXNet expects input data in the form of (batch_size, channel, height, width))</p>

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.check_input_images(),
        rule_parameters={
            "threshold_mean": "0.2",
            "threshold_samples": "500",
            "regex": ".*/hybridsequential0_input_0",
            "channel": "1"
        },
        collections_to_save=[
            CollectionConfig(
                name="custom_inputs_collection",
                parameters={
                    "include_regex": ".*/hybridsequential0_input_0"
                    "save_interval": "500"
                }
            )
        ]
    )
]

```

## NLPSequenceRatio

This rule calculates the ratio of specific tokens given the rest of the input sequence that is useful for optimizing performance. For example, you can calculate the percentage of padding end-of-sentence (EOS) tokens in your input sequence. If the number of EOS tokens is too high, an alternate bucketing strategy should be performed. You also can calculate the percentage of unknown tokens in your input sequence. If the number of unknown words is too high, an alternate vocabulary could be used.

This rule is applicable to deep learning applications.

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

### Parameter Descriptions for the NLPSequenceRatio Rule

Parameter Name	Description
base_trial	The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.

Parameter Name	Description
	<b>Required</b>  Valid values: String
<code>tensor_regex</code>	A list of regex patterns used to restrict this comparison to specific scalar-valued tensors. The rule inspects only the tensors that match the regex patterns specified in the list. If no patterns are passed, the rule compares all tensors gathered in the trials by default. Only scalar-valued tensors can be matched.  <b>Optional</b>  Valid values: List of strings or a comma-separated string  Default value: <code>".*embedding0_input_0"</code> (assuming an embedding as the initial layer of the network)
<code>token_values</code>	A string of a list of the numerical values of the tokens. For example, "3, 0".  <b>Optional</b>  Valid values: Comma-separated string of numerical values  Default value: 0
<code>token_thresholds_percent</code>	A string of a list of thresholds (in percentages) that correspond to each of the <code>token_values</code> . For example, "50.0, 50.0".  <b>Optional</b>  Valid values: Comma-separated string of floats  Default value: "50"

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.nlp_sequence_ratio(),
        rule_parameters={
            "tensor_regex": ".*embedding0_input_0",
            "token_values": "0",
            "token_thresholds_percent": "50"
        },
        collections_to_save=[
            CollectionConfig(
                name="custom_inputs_collection",
                parameters={
                    "include_regex": ".*embedding0_input_0"
                }
            )
        ]
)

```

]

## Confusion

This rule evaluates the goodness of a confusion matrix for a classification problem.

It creates a matrix of size `category_no*category_no` and populates it with data coming from `(labels, predictions)` pairs. For each `(labels, predictions)` pair, the count in `confusion[labels][predictions]` is incremented by 1. When the matrix is fully populated, the ratio of data on-diagonal values and off-diagonal values are evaluated as follows:

- For elements on the diagonal: `confusion[i][i]/sum_j(confusion[j][j])>=min_diag`
- For elements off the diagonal: `confusion[j][i])/sum_j(confusion[j][i])<=max_off_diag`

This rule can be applied to the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

### Parameter Descriptions for the Confusion Rule

Parameter Name	Description
<code>base_trial</code>	The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.  <b>Required</b>  Valid values: String
<code>category_no</code>	The number of categories.  <b>Optional</b>  Valid values: Integer $\geq 2$  Default value: "None"
<code>labels</code>	The <code>labels</code> tensor collection or an 1-d vector of true labels.  <b>Optional</b>  Valid values: String  Default value: "labels"
<code>predictions</code>	The <code>predictions</code> tensor collection or an 1-d vector of estimated labels.  <b>Optional</b>  Valid values: String  Default value: "predictions"
<code>labels_collection</code>	The rule inspects the tensors in this collection for labels.

Parameter Name	Description
	<b>Optional</b> Valid values: String Default value: "labels"
<code>predictions_collection</code>	The rule inspects the tensors in this collection for predictions. <b>Optional</b> Valid values: String Default value: "predictions"
<code>min_diag</code>	The minimum threshold for the ratio of data on the diagonal. <b>Optional</b> Valid values: <code>0≤float≤1</code> Default value: <code>0 . 9</code>
<code>max_off_diag</code>	The maximum threshold for the ratio of data off the diagonal. <b>Optional</b> Valid values: <code>0≤float≤1</code> Default value: <code>0 . 1</code>

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.confusion(),
        rule_parameters={
            "category_no": "10",
            "labels": "labels",
            "predictions": "predictions",
            "labels_collection": "labels",
            "predictions_collection": "predictions",
            "min_diag": "0.9",
            "max_off_diag": "0.1"
        },
        collections_to_save=[
            CollectionConfig(
                name="labels",
                parameters={
                    "save_interval": "500"
                }
            ),
            CollectionConfig(
                name="predictions",
                parameters={
                    "include_regex": "500"
                }
            )
        ]
)

```

**Note**

This rule infers default values for the optional parameters if their values aren't specified.

## FeatureImportanceOverweight

This rule accumulates the weights of the *n* largest feature importance values per step and ensures that they do not exceed the threshold. For example, you can set the threshold for the top 3 features to not hold more than 80 percent of the total weights of the model.

This rule is valid only for the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

### Parameter Descriptions for the FeatureImportanceOverweight Rule

Parameter Name	Description
base_trial	The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.  <b>Required</b>  Valid values: String
threshold	Defines the threshold for the proportion of the cumulative sum of the <i>n</i> largest features. The number <i>n</i> is defined by the nfeatures parameter.  <b>Optional</b>  Valid values: Float  Default value: 0 . 8
nfeatures	The number of largest features.  <b>Optional</b>  Valid values: Integer  Default value: 3
tensor_regex	Regular expression (regex) of tensor names the rule to analyze.  <b>Optional</b>  Valid values: String  Default value: ".*feature_importance/weight"

```
built_in_rules = [
```

```

Rule.sagemaker(
    base_config=rule_configs.feature_importance_overweight(),
    rule_parameters={
        "threshold": "0.8",
        "nfeatures": "3",
        "tensor_regex": ".*feature_importance/weight"
    },
    collections_to_save=[
        CollectionConfig(
            name="feature_importance",
            parameters={
                "save_interval": "500"
            }
        )
    ]
)
]

```

## TreeDepth

This rule measures the depth of trees in an XGBoost model. XGBoost rejects splits if they do not improve loss. This regularizes the training. As a result, the tree might not grow as deep as defined by the `depth` parameter.

This rule is valid only for the XGBoost algorithm.

For an example of how to configure and deploy a built-in rule, see [Configure Debugger Built-in Rules \(p. 1608\)](#).

### Parameter Descriptions for the TreeDepth Rule

Parameter Name	Description
<code>base_trial</code>	The base trial training job name. This parameter is automatically set to the current training job by Amazon SageMaker Debugger.  <b>Required</b>  Valid values: String
<code>depth</code>	The depth of the tree. The depth of the tree is obtained by computing the base 2 logarithm of the largest node ID.  <b>Optional</b>  Valid values: Float  Default value: 4

```

built_in_rules = [
    Rule.sagemaker(
        base_config=rule_configs.tree_depth(),
        rule_parameters={
            "depth": "4"
        },
        collections_to_save=[
            CollectionConfig(
                name="tree",

```

```
        parameters={
            "save_interval": "500"
        }
    ]
}
```

## Create Debugger Custom Rules for Training Job Analysis

You can create custom rules to monitor your training job using the Debugger Rule APIs and the open source [smdebug Python library](#) that provide tools to build your own rule containers.

### Topics

- [Prerequisites for Creating Debugger Custom Rules \(p. 1670\)](#)
- [Use the Debugger Client Library smdebug to Create a Custom Rule Python Script \(p. 1670\)](#)
- [Use the Debugger APIs to Run Your Own Custom Rules \(p. 1671\)](#)

## Prerequisites for Creating Debugger Custom Rules

To create Debugger custom rules, you need the following prerequisites.

- [SageMaker Debugger Rule.custom API](#)
- [The open source smdebug Python library](#)
- Your own custom rule python script
- [Amazon SageMaker Debugger Registry URLs for Custom Rule Evaluators \(p. 1739\)](#)

## Use the Debugger Client Library smdebug to Create a Custom Rule Python Script

The `smdebug` Rule API provides an interface to set up your own custom rules. The following python script is a sample of how to construct a custom rule, `CustomGradientRule`. This tutorial custom rule watches if the gradients are getting too large and set the default threshold as 10. The custom rule takes a base trial created by a SageMaker estimator when it initiates training job.

```
from smdebug.rules.rule import Rule

class CustomGradientRule(Rule):
    def __init__(self, base_trial, threshold=10.0):
        super().__init__(base_trial)
        self.threshold = float(threshold)

    def invoke_at_step(self, step):
        for tname in self.base_trial.tensor_names(collection="gradients"):
            t = self.base_trial.tensor(tname)
            abs_mean = t.reduction_value(step, "mean", abs=True)
            if abs_mean > self.threshold:
                return True
        return False
```

You can add multiple custom rule classes as many as you want in the same python script and deploy them to any training job trials by constructing custom rule objects in the following section.

## Use the Debugger APIs to Run Your Own Custom Rules

The following code sample shows how to configure a custom rule with the [Amazon SageMaker Python SDK](#). This example assumes that the custom rule script you created in the previous step is located at '`path/to/my_custom_rule.py`'.

```
from sagemaker.debugger import Rule, CollectionConfig

custom_rule = Rule.custom(
    name='MyCustomRule',
    image_uri='759209512951.dkr.ecr.us-west-2.amazonaws.com/sagemaker-debugger-rule-evaluator:latest',
    instance_type='ml.t3.medium',
    source='path/to/my_custom_rule.py',
    rule_to_invoke='CustomGradientRule',
    collections_to_save=[CollectionConfig("gradients")],
    rule_parameters={"threshold": "20.0"}
)
```

The following list explains the Debugger `Rule.custom` API arguments.

- `name` (str): Specify a custom rule name as you want.
- `image_uri` (str): This is the image of the container that has the logic of understanding your custom rule. It sources and evaluates the specified tensor collections you save in the training job. You can find the list of open source SageMaker rule evaluator images from [Amazon SageMaker Debugger Registry URLs for Custom Rule Evaluators \(p. 1739\)](#).
- `instance_type` (str): You need to specify an instance to build a rule docker container. This spins up the instance in parallel with a training container.
- `source` (str): This is the local path or the Amazon S3 URI to your custom rule script.
- `rule_to_invoke` (str): This specifies the particular Rule class implementation in your custom rule script. SageMaker supports only one rule to be evaluated at a time in a rule job.
- `collections_to_save` (str): This specifies which tensor collections you will save for the rule to run.
- `rule_parameters` (dictionary): This accepts parameter inputs in a dictionary format. You can adjust the parameters that you configured in the custom rule script.

After you set up the `custom_rule` object, you can use it for building a SageMaker estimator for any training jobs. Specify the `entry_point` to your training script. You do not need to make any change of your training script.

```
from sagemaker.tensorflow import TensorFlow

estimator = TensorFlow(
    role=sagemaker.get_execution_role(),
    base_job_name='smdebug-custom-rule-demo-tf-keras',
    entry_point='path/to/your_training_script.py'
    train_instance_type='ml.p2.xlarge'
    ...
    # debugger-specific arguments below
    rules = [custom_rule]
)

estimator.fit()
```

For more variations and advanced examples of using Debugger custom rules, see the following example notebooks.

- Monitor your training job with Amazon SageMaker Debugger custom rules
- PyTorch iterative model pruning of ResNet and AlexNet
- Trigger Amazon CloudWatch Events using Debugger Rules to Take an Action Based on Training Status with TensorFlow

## Use Debugger with Custom Training Containers

Amazon SageMaker Debugger is available for any deep learning models that you bring to Amazon SageMaker. The Amazon CLI, SageMaker Estimator API, and the Debugger APIs enable you to use any Docker base images to build and customize containers to train your models. To use Debugger with customized containers, you need to make a minimal change to your training script to implement the Debugger hook callback and retrieve tensors from training jobs.

You need the following resources to build a customized container with Debugger.

- [Amazon SageMaker Python SDK](#)
- [The SMDebug open source client library](#)
- A Docker base image of your choice
- Your training script with a Debugger hook registered – For more information about registering a Debugger hook to your training script, see [Register Debugger Hook to Your Training Script \(p. 1672\)](#).

For an end-to-end example of using Debugger with a custom training container, see the following example notebook.

- [Build a Custom Training Container and Debug Training Jobs with Debugger](#)

### Tip

This custom container with Debugger guide is an extension of the [Adapting Your Own Training Container \(p. 2151\)](#) guide which walks you thorough how to build and push your custom training container to Amazon ECR.

## Prepare to Build a Custom Training Container

To build a docker container, the basic structure of files should look like the following:

```
### debugger_custom_container_test_notebook.ipynb           # a notebook to run python snippet
codes
### debugger_custom_container_test_folder                  # this is a docker folder
    ### your-training-script.py                          # your training script with Debugger
hook
    ### Dockerfile                                     # a Dockerfile to build your own
container
```

## Register Debugger Hook to Your Training Script

To debug your model training, you need to add a Debugger hook to your training script.

### Note

This step is required to collect model parameters (output tensors) for debugging your model training. If you only want to monitor and profile, you can skip this hook registration step and exclude the `debugger_hook_config` parameter when constructing an estimator.

The following example code shows the structure of a training script using the Keras ResNet50 model and how to pass the Debugger hook as a Keras callback for debugging. To find a complete training script, see [TensorFlow training script with SageMaker Debugger hook](#).

```
# An example of training script (your-training-script.py)
import tensorflow.compat.v2 as tf
from tensorflow.keras.applications.resnet50 import ResNet50
import smdebug.tensorflow as smd

def train(batch_size, epoch, model, hook):

    ...
    model.fit(X_train, Y_train,
               batch_size=batch_size,
               epochs=epoch,
               validation_data=(X_valid, Y_valid),
               shuffle=True,

               # smdebug modification: Pass the Debugger hook in the main() as a Keras
               callback
               callbacks=[hook])

def main():
    parser=argparse.ArgumentParser(description="Train resnet50 cifar10")

    # hyperparameter settings
    parser.add_argument(...)

    args = parser.parse_args()

    model=ResNet50(weights=None, input_shape=(32,32,3), classes=10)

    # Add the following line to register the Debugger hook for Keras.
    hook=smd.KerasHook.create_from_json_file()

    # Start the training.
    train(args.batch_size, args.epoch, model, hook)

if __name__ == "__main__":
    main()
```

For more information about registering the Debugger hook for the supported frameworks and algorithm, see the following links in the SMDebug client library:

- [SMDebug TensorFlow hook](#)
- [SMDebug PyTorch hook](#)
- [SMDebug MXNet hook](#)
- [SMDebug XGBoost hook](#)

In the following example notebooks' training scripts, you can find more examples about how to add the Debugger hooks to training scripts and collect output tensors in detail:

- [Debugger in script mode with the TensorFlow 2.1 framework](#)

To see the difference between using Debugger in a Deep Learning Container and in script mode, open this notebook and put it and [the previous Debugger in a Deep Learning Container TensorFlow v2.1 notebook example](#) side by side.

In script mode, the hook configuration part is removed from the script in which you set the estimator. Instead, the Debugger hook feature is merged into the training script, [TensorFlow Keras ResNet training script in script mode](#). The training script imports the smdebug library in the required TensorFlow Keras environment to communicate with the TensorFlow ResNet50 algorithm. It also manually implements the smdebug hook functionality by adding the `callbacks=[hook]` argument

inside the `train` function (in line 49), and by adding the manual hook configuration (in line 89) provided through SageMaker Python SDK.

This script mode example runs the training job in the TF 2.1 framework for direct comparison with the zero script change in the TF 2.1 example. The benefit of setting up Debugger in script mode is the flexibility to choose framework versions not covered by Amazon Deep Learning Containers.

- [Using Amazon SageMaker Debugger in a PyTorch Container in Script Mode](#)

This notebook enables Debugger in script mode in PyTorch v1.3.1 framework. PyTorch v1.3.1 is supported by SageMaker containers, and this example shows details of how to modify a training script.

The SageMaker PyTorch estimator is already in script mode by default. In the notebook, the line to activate `script_mode` is not included in the estimator configuration.

This notebook shows detailed steps to change [an original PyTorch training script to a modified version with Debugger enabled](#). Additionally, this example shows how you can use Debugger built-in rules to detect training issues such as the vanishing gradients problem, and the Debugger trial features to call and analyze the saved tensors.

## Create and Configure a Dockerfile

Open your SageMaker JupyterLab and create a new folder, `debugger_custom_container_test_folder` in this example, to save your training script and Dockerfile. The following code example is a Dockerfile that includes essential docker build commands. Paste the following code into the Dockerfile text file and save it. Upload your training script to the same folder.

```
# Specify a docker base image
FROM tensorflow/tensorflow:2.2.0rc2-gpu-py3
RUN /usr/bin/python3 -m pip install --upgrade pip
RUN pip install --upgrade protobuf

# Install required packages to enable the SageMaker Python SDK and the smdebug library
RUN pip install sagemaker-training
RUN pip install smdebug
CMD ["bin/bash"]
```

If you want to use a pre-built Amazon Deep Learning Container image, see [Available Amazon Deep Learning Containers Images](#).

## Build and Push the Custom Training Container to Amazon ECR

Create a test notebook, `debugger_custom_container_test_notebook.ipynb`, and run the following code in the notebook cell. This will access the `debugger_byoc_test_docker` directory, build the docker with the specified `algorithm_name`, and push the docker container to your Amazon ECR.

```
import boto3

account_id = boto3.client('sts').get_caller_identity().get('Account')
ecr_repository = 'sagemaker-debugger-mnist-byoc-tf2'
tag = ':latest'

region = boto3.session.Session().region_name

uri_suffix = 'amazonaws.com'
if region in ['cn-north-1', 'cn-northwest-1']:
    uri_suffix = 'amazonaws.com.cn'
```

```
byoc_image_uri = '{}.dkr.ecr.{}/{}/{}'.format(account_id, region, uri_suffix,  
      ecr_repository + tag)  
  
!docker build -t $ecr_repository docker  
!$(aws ecr get-login --region $region --registry-ids $account_id --no-include-email)  
!aws ecr create-repository --repository-name $ecr_repository  
!docker tag {ecr_repository + tag} $byoc_image_uri  
!docker push $byoc_image_uri
```

**Tip**

If you use one of the Amazon Deep Learning Container base images, run the following code to log in to Amazon ECR and access to the Deep Learning Container image repository.

```
! aws ecr get-login-password --region {region} | docker login --username Amazon --  
password-stdin 763104351884.dkr.ecr.us-east-1.amazonaws.com
```

## Run and Debug Training Jobs Using the Custom Training Container

After you build and push your docker container to Amazon ECR, configure a SageMaker estimator with your training script and the Debugger-specific parameters. After you execute the `estimator.fit()`, Debugger will collect output tensors, monitor them, and detect training issues. Using the saved tensors, you can further analyze the training job by using the smdebug core features and tools. Configuring a workflow of Debugger rule monitoring process with Amazon CloudWatch Events and Amazon Lambda, you can automate a stopping training job process whenever the Debugger rules spots training issues.

```
import sagemaker  
from sagemaker.estimator import Estimator  
from sagemaker.debugger import Rule, DebuggerHookConfig, CollectionConfig, rule_configs  
  
profiler_config=ProfilerConfig(...)  
debugger_hook_config=DebuggerHookConfig(...)  
rules=[  
    Rule.sagemaker(rule_configs.built_in_rule()),  
    ProfilerRule.sagemaker(rule_configs.BuiltInRule())  
]  
  
estimator=Estimator(  
    image_uri=byoc_image_uri,  
    entry_point='./debugger_custom_container_test_folder/your-training-script.py'  
    role=sagemaker.get_execution_role(),  
    base_job_name='debugger-custom-container-test',  
    instance_count=1,  
    instance_type='ml.p3.2xlarge',  
  
    # Debugger-specific parameters  
    profiler_config=profiler_config,  
    debugger_hook_config=debugger_hook_config,  
    rules=rules  
)  
  
# start training  
estimator.fit()
```

## Action on Amazon SageMaker Debugger Rules

Based on the Debugger rule evaluation status, you can set up automated actions such as stopping a training job and sending notifications using Amazon Simple Notification Service (Amazon SNS). You can

also create your own actions using Amazon CloudWatch Events and Amazon Lambda. To learn how to set up automated actions based on the Debugger rule evaluation status, see the following topics.

#### Topics

- [Debugger Built-in Actions for Rules \(p. 1676\)](#)
- [Create Actions on Rules Using Amazon CloudWatch and Amazon Lambda \(p. 1680\)](#)

## Debugger Built-in Actions for Rules

Use Debugger built-in actions to respond to issues found by [Debugger Rule \(p. 1627\)](#). The `Debugger rule_configs` class provides tools to configure a list of actions, including automatically stopping training jobs and sending notifications using Amazon Simple Notification Service (Amazon SNS) when the Debugger rules find training issues.

### Step 1: Set Up Amazon SNS, Create an SMDebugRules Topic, and Subscribe to the Topic

This section walks you through how to set up an Amazon SNS `SMDebugRules` topic, subscribe to it, and confirm the subscription to receive notifications from the Debugger rules.

#### Note

For more information about billing for Amazon SNS, see [Amazon SNS pricing](#) and [Amazon SNS FAQs](#).

#### To create a SMDebugRules topic

1. Sign in to the Amazon Web Services Management Console and open the Amazon SNS console at <https://console.amazonaws.cn/sns/v3/home>.
2. In the left navigation pane, choose **Topics**.
3. On the **Topics** page, choose **Create topic**.
4. On the **Create topic** page, in the **Details** section, do the following:
  - a. For **Type**, choose **Standard** for topic type.
  - b. In **Name**, enter **SMDebugRules**.
5. Skip all other optional settings and choose **Create topic**. If you want to learn more about the optional settings, see [Creating an Amazon SNS topic](#).

#### To subscribe to the SMDebugRules topic

1. Open the Amazon SNS console at <https://console.amazonaws.cn/sns/v3/home>.
2. In the left navigation pane, choose **Subscriptions**.
3. On the **Subscriptions** page, choose **Create subscription**.
4. On the **Create subscription** page, in the **Details** section, do the following:
  - a. For **Topic ARN**, choose the **SMDebugRules** topic ARN. The ARN should be in format of `arn:aws:sns:<region-id>:111122223333:SMDebugRules`.
  - b. For **Protocol**, choose **Email** or **SMS**.
  - c. For **Endpoint**, enter the endpoint value, such as an email address or a phone number that you want to receive notifications.

#### Note

Make sure you type the correct email address and phone number. Phone numbers must include +, a country code, and phone number, with no special characters or spaces. For example, the phone number +1 (222) 333-4444 is formatted as **+12223334444**.

- Skip all other optional settings and choose **Create subscription**. If you want to learn more about the optional settings, see [Subscribing to an Amazon SNS topic](#).

After you subscribe to the **SMDebugRules** topic, you receive the following confirmation message in email or by phone:



For more information about Amazon SNS, see [Mobile text messaging \(SMS\)](#) and [Email notifications](#) in the [Amazon SNS Developer Guide](#).

## Step 2: Set Up Your IAM Role to Attach Required Policies

In this step, you add the required policies to your IAM role.

### To add the required policies to your IAM role

- Sign in to the Amazon Web Services Management Console and open the IAM console at <https://console.amazonaws.cn/iam/>.
- In the left navigation pane, choose **Policies**, and choose **Create policy**.
- On the **Create policy** page, do the following to create a new sns-access policy:
  - Choose the **JSON** tab.
  - Paste the JSON strings formatted in bold in the following code into the "Statement", replacing the 12-digit Amazon account ID with your Amazon account ID.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "VisualEditor0",  
            "Effect": "Allow",  
            "Action": [  
                "sns:Publish",  
                "sns:CreateTopic",  
                "sns:Subscribe"  
            ],  
            "Resource": "arn:aws:sns:*:111122223333:SMDebugRules"  
        }  
    ]  
}
```

- At the bottom of the page, choose **Review policy**.
  - On the **Review policy** page, for **Name**, enter **sns-access**.
  - At the bottom of the page, choose **Create policy**.
- Go back to the IAM console, and choose **Roles** in the left navigation pane.

5. Look up the IAM role that you use for SageMaker model training and choose that IAM role.
6. On the **Permissions** tab of the **Summary** page, choose **Attach policies**.
7. Search for the **sns-access** policy, select the check box next to the policy, and then choose **Attach policy**.

For more examples of setting up IAM policies for Amazon SNS, see [Example cases for Amazon SNS access control](#).

## Step 3: Configure Debugger Rules with the Built-in Actions

After successfully finishing the required settings in the preceding steps, you can configure the Debugger built-in actions for debugging rules as shown in the following example script. You can choose which built-in actions to use while building the `actions` list object. The `rule_configs` is a helper module that provides high-level tools to configure Debugger built-in rules and actions. The following built-in actions are available for Debugger:

- `rule_configs.StopTraining()` – Stops a training job when the Debugger rule finds an issue.
- `rule_configs.Email("abc@abc.com")` – Sends a notification via email when the Debugger rule finds an issue. Use the email address that you used when you set up your SNS topic subscription.
- `rule_configs.SMS("+1234567890")` – Sends a notification via text message when the Debugger rule finds an issue. Use the phone number that you used when you set up your SNS topic subscription.

### Note

Make sure you type the correct email address and phone number. Phone numbers must include +, a country code, and a phone number, with no special characters or spaces. For example, the phone number +1 (222) 333-4444 is formatted as **+12223334444**.

You can use all of the built-in actions or a subset of actions by wrapping up using the `rule_configs.ActionList()` method, which takes the built-in actions and configures a list of actions.

### To add all of the three built-in actions to a single rule

If you want to assign all of the three built-in actions to a single rule, configure a Debugger built-in action list while constructing an estimator. Use the following template to construct the estimator, and Debugger will stop training jobs and send notifications through email and text for any rules that you use to monitor your training job progress.

```
from sagemaker.debugger import Rule, rule_configs

# Configure an action list object for Debugger rules
actions = rule_configs.ActionList(
    rule_configs.StopTraining(),
    rule_configs.Email("abc@abc.com"),
    rule_configs.SMS("+1234567890")
)

# Configure rules for debugging with the actions parameter
rules = [
    Rule.sagemaker(
        base_config=rule_configs.built_in_rule(),           # Required
        rule_parameters={"paramter_key": "value"},          # Optional
        actions=actions
    )
]

estimator = Estimator(
    ...
)
```

```

        rules = rules
    )

estimator.fit(wait=False)

```

### To create multiple built-in action objects to assign different actions to a single rule

If you want to assign the built-in actions to be triggered at different threshold values of a single rule, you can create multiple built-in action objects as shown in the following script. To avoid a conflict error by running the same rule, you must submit different rule job names (specify different strings for the rules' name attribute) as shown in the following example script template. This example shows how to set up [StalledTrainingRule](#) (p. 1658) to take two different actions: send an email to abc@abc.com when a training job stalls for 60 seconds, and stop the training job if stalling for 120 seconds.

```

from sagemaker.debugger import Rule, rule_configs
import time

base_job_name_prefix= 'smdebug-stalled-demo-' + str(int(time.time()))

# Configure an action object for StopTraining
action_stop_training = rule_configs.ActionList(
    rule_configs.StopTraining()
)

# Configure an action object for Email
action_email = rule_configs.ActionList(
    rule_configs.Email("abc@abc.com")
)

# Configure a rule with the Email built-in action to trigger if a training job stalls for
# 60 seconds
stalled_training_job_rule_email = Rule.sagemaker(
    base_config=rule_configs.stalled_training_rule(),
    rule_parameters={
        "threshold": "60",
        "training_job_name_prefix": base_job_name_prefix
    },
    actions=action_email
)
stalled_training_job_rule_email.name="StalledTrainingJobRuleEmail"

# Configure a rule with the StopTraining built-in action to trigger if a training job
# stalls for 120 seconds
stalled_training_job_rule = Rule.sagemaker(
    base_config=rule_configs.stalled_training_rule(),
    rule_parameters={
        "threshold": "120",
        "training_job_name_prefix": base_job_name_prefix
    },
    actions=action_stop_training
)
stalled_training_job_rule.name="StalledTrainingJobRuleStopTraining"

estimator = Estimator(
    ...
    rules = [stalled_training_job_rule_email, stalled_training_job_rule]
)

estimator.fit(wait=False)

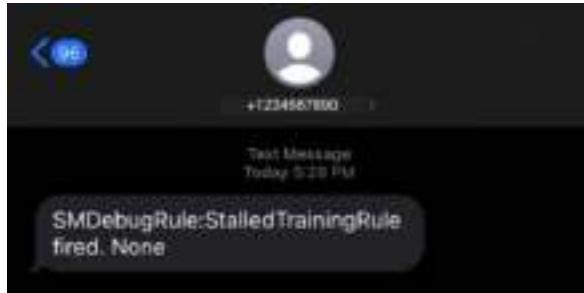
```

While the training job is running, the Debugger built-in action sends notification emails and text messages whenever the rule finds issues with your training job. The following screenshot shows an example of email notification for a training job that has a stalled training job issue.

## SMDebugRule:StalledTrainingRule fired

S SMDestroyRules <no-reply@sns.amazonaws.com> Today at 1:35 PM  
To:  
  
SMDebugRule:StalledTrainingRule fired. None  
  
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:  
<https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:111122223333:SMDestroyRules:c6e0d93c-435a-4c43-084b-d98b4f12b19c&Endpoint>  
Please do not reply directly to this email. If you have any questions or comments regarding this email, please contact us at <https://aws.amazon.com/support>

The following screenshot shows an example text notification that Debugger sends when the rule finds a StalledTraining issue.



## Considerations for Using the Debugger Built-in Actions

- To use the Debugger built-in actions, an internet connection is required. This feature is not supported in the network isolation mode provided by Amazon SageMaker or Amazon VPC.
- The built-in actions cannot be used for [Debugger ProfilerRule \(p. 1626\)](#).
- The built-in actions cannot be used on training jobs with spot training interruptions.
- In email or text notifications, None appears at the end of messages. This does not have any meaning, so you can disregard the text None.

## Create Actions on Rules Using Amazon CloudWatch and Amazon Lambda

Amazon CloudWatch collects Amazon SageMaker model training job logs and Amazon SageMaker Debugger rule processing job logs. Configure Debugger with Amazon CloudWatch Events and Amazon Lambda to take action based on Debugger rule evaluation status.

### CloudWatch Logs for Debugger Rules and Training Jobs

#### To find training job logs and Debugger rule job logs

1. Open the CloudWatch console at <https://console.amazonaws.cn/cloudwatch/>.
2. In the left navigation pane under the **Log** node, choose **Log Groups**.
3. In the log groups list, do the following:
  - Choose **/aws/sagemaker/TrainingJobs** for training job logs.

- Choose **/aws/sagemaker/ProcessingJobs** for Debugger rule job logs.

You can use the training and Debugger rule job status in the CloudWatch logs to take further actions when there are training issues.

For more information about monitoring training jobs using CloudWatch, see [Monitor Amazon SageMaker](#).

## Set Up Debugger for Automated Training Job Termination Using CloudWatch and Lambda

The Debugger rules monitor training job status, and a CloudWatch Events rule watches the Debugger rule training job evaluation status.

### Step 1: Create a Lambda Function

#### To create a Lambda function

1. Open the Amazon Lambda console at <https://console.amazonaws.cn/lambda/>.
2. In the left navigation pane, choose **Functions** and then choose **Create function**.
3. On the **Create function** page, choose **Author from scratch** option.
4. In the **Basic information** section, enter a **Function name** (for example, **debugger-rule-stop-training-job**).
5. For **Runtime**, choose **Python 3.7**.
6. For **Permissions**, expand the drop down option, and choose **Change default execution role**.
7. For **Execution role**, choose **Use an existing role** and choose the IAM role that you use for training jobs on SageMaker.

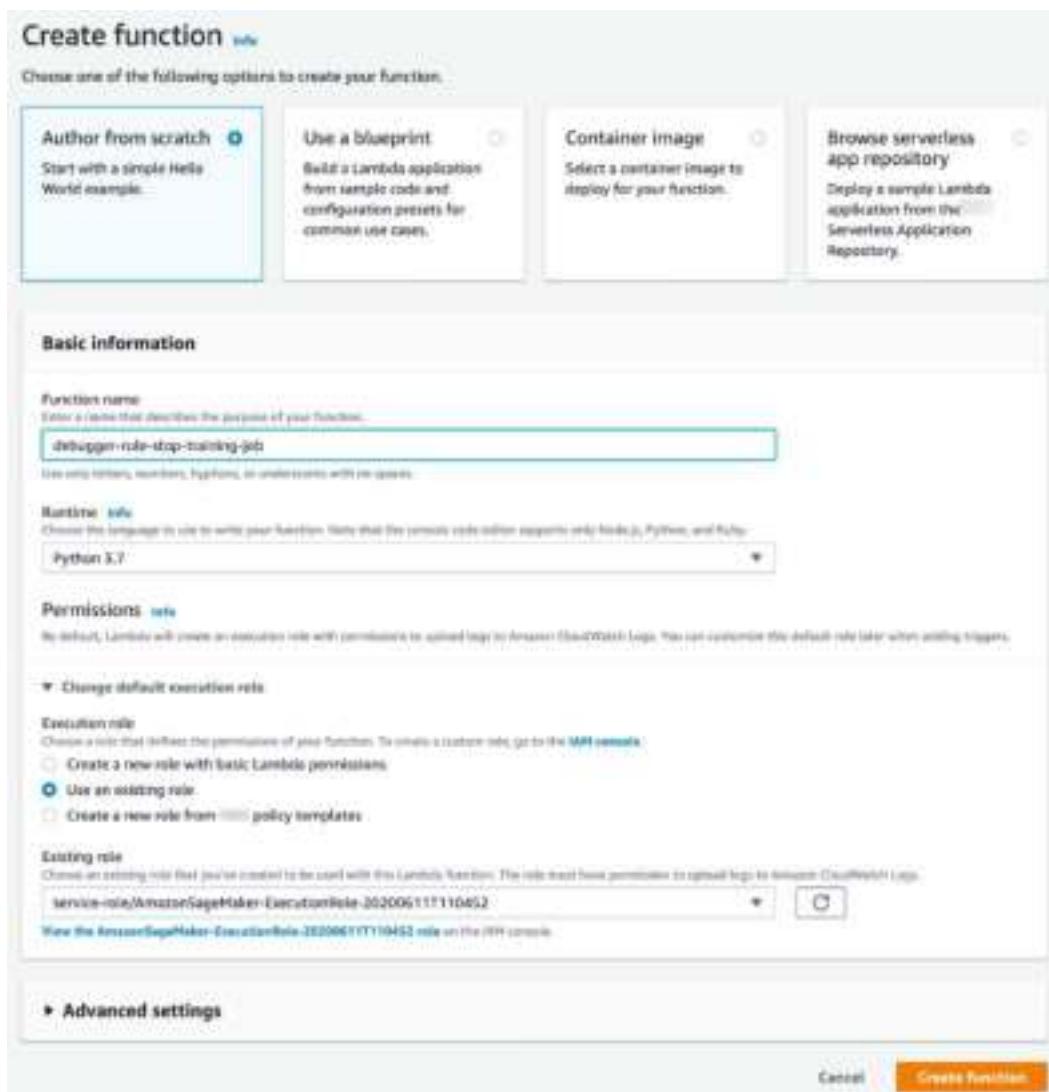
#### Note

Make sure you use the execution role with `AmazonSageMakerFullAccess` and `AWSLambdaBasicExecutionRole` attached. Otherwise, the Lambda function won't properly react to the Debugger rule status changes of the training job. If you are unsure which execution role is being used, run the following code in a Jupyter notebook cell to retrieve the execution role output:

```
import sagemaker
sagemaker.get_execution_role()
```

8. At the bottom of the page, choose **Create function**.

The following figure shows an example of the **Create function** page with the input fields and selections completed.



## Step 2: Configure the Lambda function

### To configure the Lambda function

1. In the **Function code** section of the configuration page, paste the following Python script in the Lambda code editor pane. The `lambda_handler` function monitors the Debugger rule evaluation status collected by CloudWatch and triggers the `StopTrainingJob` API operation. The Amazon SDK for Python (Boto3) client for SageMaker provides a high-level method, `stop_training_job`, which triggers the `StopTrainingJob` API operation.

```

import json
import boto3
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    training_job_name = event.get("detail").get("TrainingJobName")
    logging.info(f'Evaluating Debugger rules for training job: {training_job_name}')

```

```

eval_statuses = event.get("detail").get("DebugRuleEvaluationStatuses", None)

if eval_statuses is None or len(eval_statuses) == 0:
    logging.info("Couldn't find any debug rule statuses, skipping...")
    return {
        'statusCode': 200,
        'body': json.dumps('Nothing to do')
    }

# should only attempt stopping jobs with InProgress status
training_job_status = event.get("detail").get("TrainingJobStatus", None)
if training_job_status != 'InProgress':
    logging.debug(f"Current Training job status({training_job_status}) is not
'InProgress'. Exiting")
    return {
        'statusCode': 200,
        'body': json.dumps('Nothing to do')
    }

client = boto3.client('sagemaker')

for status in eval_statuses:
    logging.info(status.get("RuleEvaluationStatus") + ', RuleEvaluationStatus=' +
str(status))
    if status.get("RuleEvaluationStatus") == "IssuesFound":
        secondary_status = event.get("detail").get("SecondaryStatus", None)
        logging.info(
            f'About to stop training job, since evaluation of rule
configuration {status.get("RuleConfigurationName")}) resulted in "IssuesFound". ' +
            f'\ntraining job "{training_job_name}" status is
"{training_job_status}", secondary status is "{secondary_status}"' +
            f'\nAttempting to stop training job "{training_job_name}"'
        )
    )
try:
    client.stop_training_job(
        TrainingJobName=training_job_name
    )
except Exception as e:
    logging.error(
        "Encountered error while trying to "
        "stop training job {}: {}".format(
            training_job_name, str(e)
        )
    )
    raise e
return None

```

For more information about the Lambda code editor interface, see [Creating functions using the Amazon Lambda console editor](#).

2. Skip all other settings and choose **Save** at the top of the configuration page.

### Step 3: Create a CloudWatch Events Rule and Link to the Lambda Function for Debugger

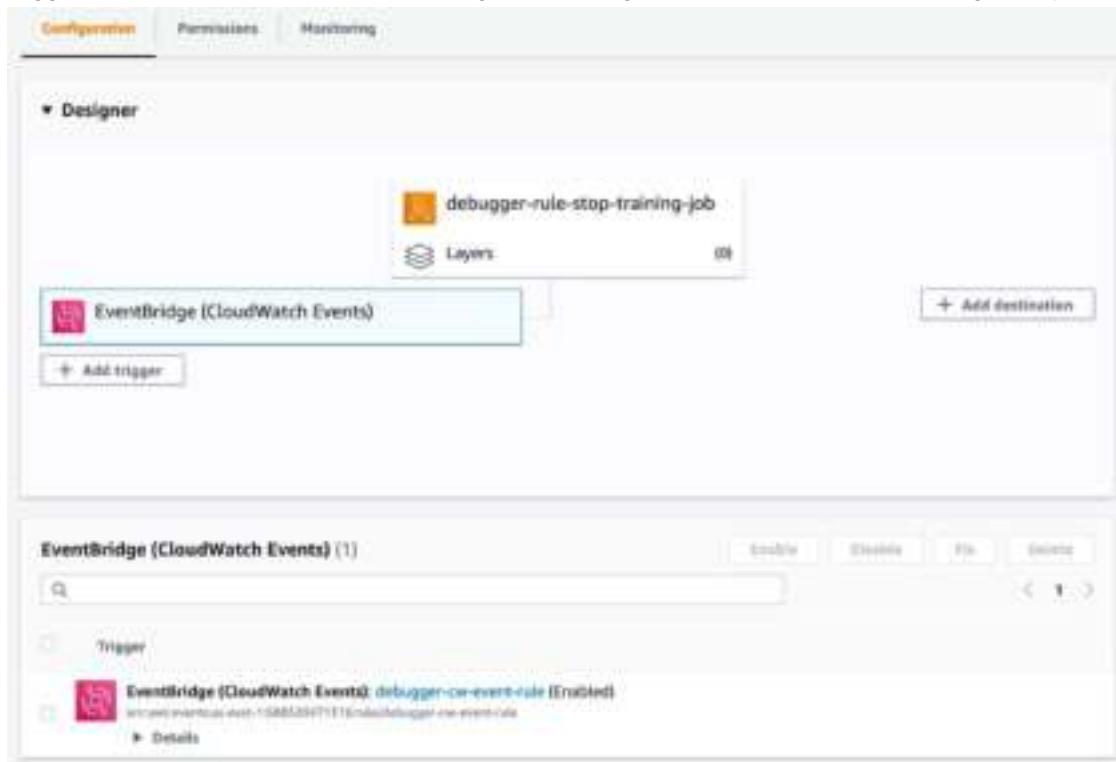
#### To create a CloudWatch Events rule and link to the Lambda function for Debugger

1. Open the CloudWatch console at <https://console.amazonaws.cn/cloudwatch/>.
2. In the left navigation pane, choose **Rules** under the **Events** node.
3. Choose **Create rule**.

4. In the **Event Source** section of the **Step 1: Create rule** page, choose **SageMaker** for **Service Name**, and choose **SageMaker Training Job State Change** for **Event Type**. The Event Pattern Preview should look like the following example JSON strings:

```
{  
    "source": [  
        "aws.sagemaker"  
    ],  
    "detail-type": [  
        "SageMaker Training Job State Change"  
    ]  
}
```

5. In the **Targets** section, choose **Add target\***, and choose the **debugger-rule-stop-training-job** Lambda function that you created. This step links the CloudWatch Events rule with the Lambda function.
6. Choose **Configure details** and go to the **Step 2: Configure rule details** page.
7. Specify the CloudWatch rule definition name. For example, **debugger-cw-event-rule**.
8. Choose **Create rule** to finish.
9. Go back to the Lambda function configuration page and refresh the page. Confirm that it's configured correctly in the **Designer** panel. The CloudWatch Events rule should be registered as a trigger for the Lambda function. The configuration design should look like the following example:



## Run Example Notebooks to Test Automated Training Job Termination

You can run the following example notebooks, which are prepared for experimenting with stopping a training job using Debugger's built-in rules.

- [Amazon SageMaker Debugger - Reacting to CloudWatch Events from Debugger rules](#)

This example notebook runs a training job that has a vanishing gradient issue. The Debugger [VanishingGradient \(p. 1646\)](#) built-in rule is used while constructing the SageMaker TensorFlow estimator. When the Debugger rule detects the issue, the training job is terminated.

- [Detect stalled training and stop training job using Debugger rule](#)

This example notebook runs a training script with a code line that forces it to sleep for 10 minutes. The Debugger [StalledTrainingRule \(p. 1658\)](#) built-in rule invokes issues and stops the training job.

## Disable the CloudWatch Events Rule to Stop Using the Automated Training Job Termination

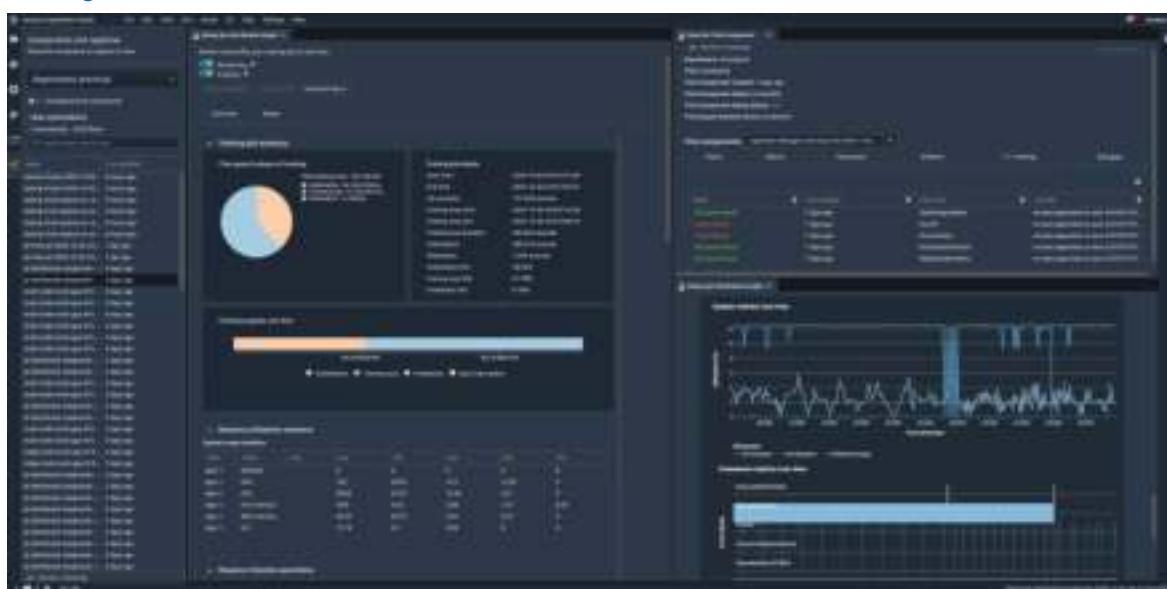
If you want to disable the automated training job termination, you need to disable the CloudWatch Events rule. In the Lambda **Designer** panel, choose the **EventBridge (CloudWatch Events)** block linked to the Lambda function. This shows an **EventBridge** panel below the **Designer** panel (for example, see the [previous screen shot](#)). Select the check box next to **EventBridge (CloudWatch Events): debugger-cw-event-rule**, and then choose **Disable**. If you want to use the automated termination functionality later, you can enable the CloudWatch Events rule again.

## SageMaker Debugger on Studio

Use Debugger dashboards on Studio to analyze your model performance and system bottlenecks while running training jobs on Amazon Elastic Compute Cloud instances. Gain insights into your training jobs and improve your model training performance and accuracy with the Debugger dashboards. By default, Debugger monitors system metrics (CPU, GPU, CPU and GPU memory, Network, and data I/O) every 500 milliseconds and basic output tensors (loss and accuracy) every 500 iterations for training jobs. You can also further customize Debugger configuration parameter values and adjust the saving intervals through the Studio UI or using the [Amazon SageMaker Python SDK](#).

### Important

If using existing SageMaker Studio apps, you need to restart them to use the new features. For instructions on how to restart and update your Studio environment, see [Update Amazon SageMaker Studio](#).



### Topics

- [Open Amazon SageMaker Debugger Insights Dashboard \(p. 1686\)](#)

- [SageMaker Debugger Insights Dashboard Controller \(p. 1687\)](#)
- [SageMaker Debugger Insights Dashboard Walkthrough \(p. 1691\)](#)
- [Shut Down the SageMaker Debugger Insights Instance \(p. 1697\)](#)
- [SageMaker Debugger on Studio Experiments \(p. 1698\)](#)

## Open Amazon SageMaker Debugger Insights Dashboard

Open the Debugger insights dashboard on Studio to see profiling progress and results of resource utilization and system bottlenecks of your training job running on EC2 instances.

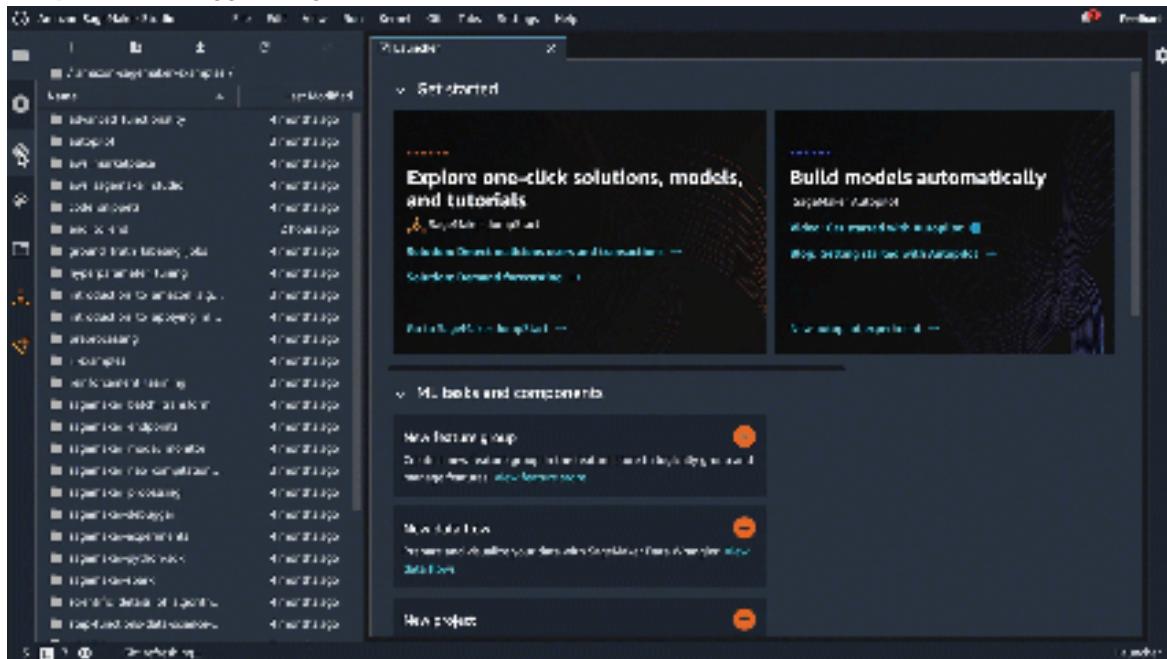
### Note

Studio Debugger insights dashboard runs a Studio app on an `m1.m5.4xlarge` instance to process and render the visualizations. Each Debugger insights tab runs one Studio kernel session. Multiple kernel sessions for multiple Debugger insights tabs run on the single instance. When you close a Debugger insights tab, the corresponding kernel session is also closed. A Studio app remains active and accrue charges for the `m1.m5.4xlarge` instance usage. For information about pricing, see the [Amazon SageMaker Pricing](#) page.

### Important

When you are done using the Debugger insights dashboard, you must shut down the `m1.m5.4xlarge` instance to avoid accruing charges. For instructions on how to shut down the instance, see [Shut Down the SageMaker Debugger Insights Instance \(p. 1697\)](#).

### To open the Debugger insights dashboard



1. Choose the **SageMakerComponents and registries** icon (💡).
2. Open the drop down menu, and choose **Experiments and trials**.
3. Look up your training job name. If you have not assigned a SageMaker Experiments trial component to the training job, the job is collected under the **Unassigned trial components** list.
4. **Right-click** (or an equivalent UI interaction) to open the context menu of the training job trial component. There are two menu items to access the Debugger features on Studio: **Open Debugger for insights** and **Open in trial details**.

5. Choose **Open Debugger for insights**. This opens a **Debug [your-training-job-name]** tab. In this tab, Debugger provides an overview of your model training performance on EC2 instances and identifies system bottleneck problems. While **monitoring** the system resource utilization, you can also enable **profiling** to capture framework metrics that consist of data from neural network operations executed during forward and backward pass and data loading. For more information about how to enable **profiling** using the Debugger insights dashboard controller, see [Enable and Configure Debugger Profiling for Detailed Insights \(p. 1688\)](#).

Debugger correlates the system resource utilization metrics with the framework metrics and helps identify resource-intensive operators that might be the root cause of the system bottlenecks. You can also download an aggregated Debugger profiling report. For more information, see [SageMaker Debugger Insights Dashboard Controller \(p. 1687\)](#).

## SageMaker Debugger Insights Dashboard Controller

There are different components of the Debugger Controller for monitoring and profiling. In this guide, you learn about the Debugger controller components.

### Note

Studio Debugger insights dashboard runs a Studio app on an `m1.m5.4xlarge` instance to process and render the visualizations. Each Debugger insights tab runs one Studio kernel session. Multiple kernel sessions for multiple Debugger insights tabs run on the single instance. When you close a Debugger insights tab, the corresponding kernel session is also closed. A Studio app remains active and accrue charges for the `m1.m5.4xlarge` instance usage. For information about pricing, see the [Amazon SageMaker Pricing](#) page.

### Important

When you are done using the Debugger insights dashboard, you must shut down the `m1.m5.4xlarge` instance to avoid accruing charges. For instructions on how to shut down the instance, see [Shut Down the SageMaker Debugger Insights Instance \(p. 1697\)](#).

## SageMaker Debugger Insights Controller UI

Using the Debugger controller located at the upper-left corner of the insights dashboard, you can refresh the dashboard, configure or update Debugger settings for monitoring system metrics and profiling framework metrics, stop training job, and download Debugger profiling report.



- If you want to manually refresh the **Debug [your-training-job-name]** page, choose the refresh button (the round arrow at the upper-left corner) as shown in the preceding screenshot.
- **Monitoring** is on by default for any SageMaker training job. By monitoring your training job, Debugger only collects system metrics to detect resource utilization problems, such as CPU bottlenecks and GPU underutilization. For a complete list of resource utilization problems that Debugger monitors, see [Debugger Built-in Rules for Monitoring Hardware System Resource Utilization \(System Metrics\) \(p. 1626\)](#).

- To download a comprehensive Debugger profiling report with details and analysis of a training job, choose **Download report**. For more information about the Debugger profiling report, see [SageMaker Debugger Profiling Report \(p. 1701\)](#).

## Enable and Configure Debugger Profiling for Detailed Insights

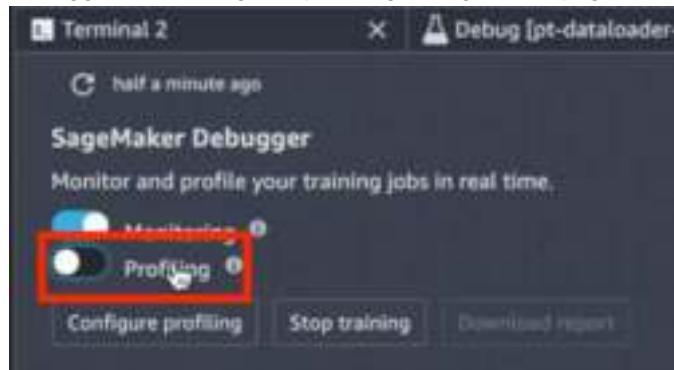
When you enable **Profiling**, Debugger starts collecting framework metrics. Framework metrics are model data collected from ML framework operations of your model, such as forward pass, backward pass, batch normalization, and data loader processes. Debugger correlates system performance bottlenecks with the framework operations and runs the [Debugger Built-in Rules for Profiling Framework Metrics \(p. 1626\)](#).

### Note

After you've enabled profiling, Debugger collects every framework operation call that's executed in each step: operations for convolving down input layers during forward pass, updating weights of millions of neurons during backward pass, and data loader processes. While profiling enables you to understand model performance at a deeper level, collecting framework metrics might impact your training time and performance. We recommend that you enable profiling to inspect your model up to two steps at a time. For more information about how to configure Debugger for framework profiling using [Amazon SageMaker Python SDK](#), see [Configure Debugger Framework Profiling \(p. 1598\)](#) and [Updating Debugger System Monitoring and Framework Profiling Configuration while a Training Job is Running \(p. 1602\)](#).

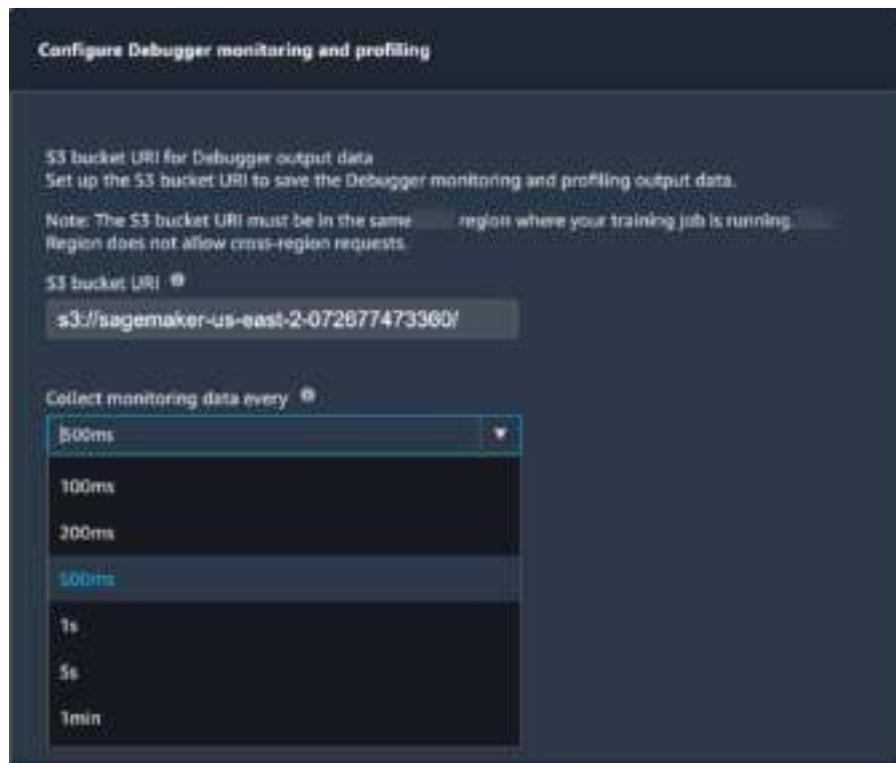
If you want to enable profiling while your training job is running, use the following steps to start profiling.

1. In SageMaker Studio, turn on **Profiling** to enable Debugger framework profiling. This opens a Debugger monitoring and profiling configuration page.



2. In **Configure Debugger monitoring and profiling**, **S3 bucket URI** and **Collect monitoring data every** are already set to the default values.

You can choose to change the monitoring interval using the drop down menu and selecting among the following available options: 100 milliseconds, 200 milliseconds, 500 milliseconds (default), 1 second, 5 seconds, and 1 minute.



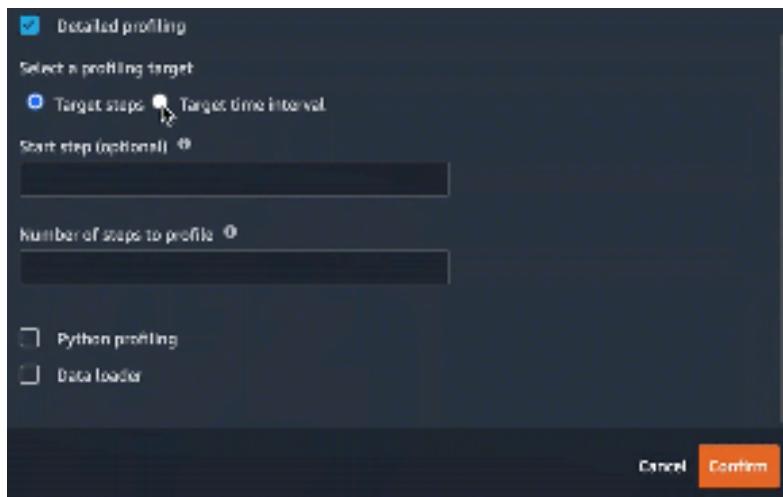
The following fields need to be specified:

- S3 bucket URI – Specify the base S3 bucket URI.
- Collect monitoring data every – Select a time interval to collect system metrics.

**Note**

If you choose one of the lower time intervals, you increase the granularity of monitoring system metrics. It allows you to capture spikes and anomalies with a higher time resolution. However, as the size of system metrics to process proportionally increases, it might impact the overall training and processing time.

3. In **Advanced settings for profiling**, configure framework metrics profiling options. Specify **Start step** (or **Start time**) and **Number of steps to profile** (or **Time duration to profile**) to profile. You can also leave the input fields blank. The default values will be automatically configured to use the current step and for 1 step duration.



- **Detailed profiling** – Specify a target step or time range to profile framework operations using the native framework profilers (TensorFlow profiler and PyTorch profiler). For example, if using TensorFlow, the Debugger hooks enable the TensorFlow profiler to collect TensorFlow-specific framework metrics. Detailed profiling enables you to profile all framework operators at a pre-step (before the first step), within steps, and between steps of a training job.

**Note**

The detailed profiling might significantly increase GPU memory consumption. It is not recommended to enable the detailed profiling for more than a couple of steps.

- **Python profiling** – Specify a target step or time range to profile Python functions. You can also choose between two Python profilers: cProfile and Pyinstrument.

- **cProfile** – The standard Python profiler. cProfile collects for every Python operator called during training. With cProfile, Debugger saves cumulative time and annotation of each function call, providing a complete detail of Python functions. In deep learning, for example, the most frequently called functions might be the convolutional filters and backward pass operators, and cProfile profiles every single of them. For the cProfile option, you can further select a timer option: total time, CPU time, and off-CPU time. While you can profile every function calls executing on processors (both CPU and GPU) in CPU time, you can also identify I/O or network bottlenecks with the off-CPU time option. The default is total time, and Debugger profiles both CPU and off-CPU time. With cProfile, you are able to drill down to every single functions when analyzing the profile data.
- **Pyinstrument** – Pyinstrument is a low overhead Python profiler that works based on sampling. With the Pyinstrument option, Debugger samples profiling events every millisecond. Because Pyinstrument measures elapsed wallclock time instead of CPU time, the Pyinstrument option can be a better choice over the cProfile option for reducing profiling noises (filtering out irrelevant function calls that are cumulatively fast) and capturing operators that are actually compute intensive (cumulatively slow) for training your model. With Pyinstrument, you are able to see a tree of function calls and better understand the structure and root cause of the slowness.

**Note**

Enabling Python profiling might result in slowing down the overall training time. In case of cProfile, Python operators that are the most frequently called are profiled at every call, so the processing time on profiling increases with respect to the number of calls. For Pyinstrument, the cumulative profiling time increases with respect to time because of its sampling mechanism.

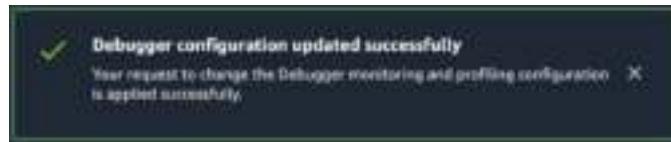
- **Dataloader profiling** – Specify a target step or time range to profile deep learning framework data loader processes. Debugger collects every data loader event of the frameworks.

**Note**

The data loader profiling can lower the training performance while collecting information from data loaders. We don't recommend that you enable the data loader profiling for more than a couple of steps.

Debugger is pre-configured to annotate data loader processes only for the Amazon deep learning containers. Debugger cannot profile data loader processes on any other custom or external containers.

4. Choose **Confirm** to finish your profiling configuration. When the configuration is successfully updated, you should be able to see the following confirmation message.



## SageMaker Debugger Insights Dashboard Walkthrough

When you initiate a SageMaker training job, Debugger starts monitoring hardware system resource utilization of EC2 instances by default. You can track the system utilization rate, statistics overview, and bottleneck detection status and results through the Studio. This guide walks you through every component of the Studio Debugger insights dashboard.

**Note**

Studio Debugger insights dashboard runs a Studio app on an `m1.m5.4xlarge` instance to process and render the visualizations. Each Debugger insights tab runs one Studio kernel session. Multiple kernel sessions for multiple Debugger insights tabs run on the single instance. When you close a Debugger insights tab, the corresponding kernel session is also closed. A Studio app remains active and accrue charges for the `m1.m5.4xlarge` instance usage. For information about pricing, see the [Amazon SageMaker Pricing](#) page.

**Important**

When you are done using the Debugger insights dashboard, you must shut down the `m1.m5.4xlarge` instance to avoid accruing charges. For instructions on how to shut down the instance, see [Shut Down the SageMaker Debugger Insights Instance \(p. 1697\)](#).

**Topics**

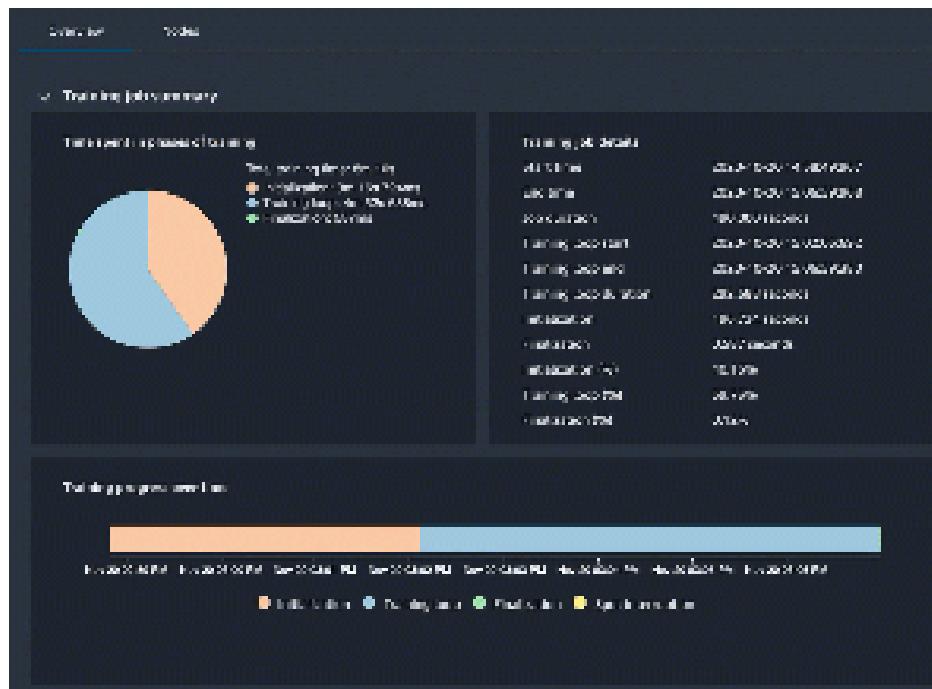
- [Debugger Insights – Overview \(p. 1691\)](#)
- [Debugger Insights – Nodes \(p. 1694\)](#)

### Debugger Insights – Overview

On the **Overview** tab, you can find a training job summary, resource utilization summary, resource intensive operations, and insights.

#### Training job summary

The **Training job summary** section shows the overall training time spent on different phases of training: initialization, training loop, and finalization. The pie chart shows the time usage percentage and absolute time amount spent on the different training phases. For example, you can have a high-level overview of how long it takes for initializing a training job, check if the initialization is taking too long due to data downloading, leaving the GPUs idle.



This section has the following features:

- The **Training progress over time** chart shows the timeline of the different training phase over time. If using spot training, you can also find the spot interruptions in the timeline chart.
- The **Training job details** panel shows the exact time stamps and utilization rate percentage numbers.
  - **Start time** – The exact time when the training job started.
  - **End time** – The exact time when the training job finished.
  - **Job duration** – The total training time from the **Start time** to the **End time**.
  - **Training loop start** – The exact time when the first step of the first epoch has started.
  - **Training loop end** – The exact time when the last step of the last epoch has finished.
  - **Training loop duration** – The total time between the Training loop start time and the Training loop end time.
  - **Initialization** – Time spent on initializing the training job, such as compiling the training script, initiating EC2 instances, and downloading training data.
  - **Finalization** – Time spent on finalizing the training job, such as finishing the model training, updating the model artifacts, and closing the EC2 instances.
  - **Initialization (%)** – The percentage of time spent on **Initialization** over the total **Job duration**.
  - **Training loop (%)** – The percentage of time spent on **Training loop** over the total **Job duration**.
  - **Finalization (%)** – The percentage of time spent on **Finalization** over the total **Job duration**.

### Resource utilization summary

This summary table shows hardware system resource utilization statistics of all workers (algo-n). System metrics include total CPU utilization, total GPU utilization, total CPU memory utilization, total GPU memory utilization, total I/O wait time, and total Network in bytes. The table shows the minimum and the maximum values, and p99, p90, and p50 percentiles.

Resource utilization summary							
System usage statistics							
Node	Name	CPU	Mem	GPU	Mem	GPU	Mem
sage-1	Network	344956425.3	0	0	0	0	0
sage-2	Network	0	0	0	0	0	0
sage-1	GPU	79	74.29	68.75	0	0	0
sage-2	GPU	75	74.25	66	0	0	0
sage-1	CPU	44.00	11.78	10.11	0	0	0
sage-2	CPU	37.5	11.87	10.43	0	0	0
sage-1	CPU memory	0.00	0.00	0.00	0.00	0.00	0.00
sage-2	CPU memory	0	0	0.00	0.00	0.00	0.00
sage-1	CPU memory	54.75	14.25	13.10	0	0	0
sage-2	CPU memory	67.75	11.75	14.5	0	0	0
sage-1	GPU	28.16	5.22	8.27	0	0	0
sage-2	GPU	23.00	4.25	9.02	0	0	0

## Resource intensive operations

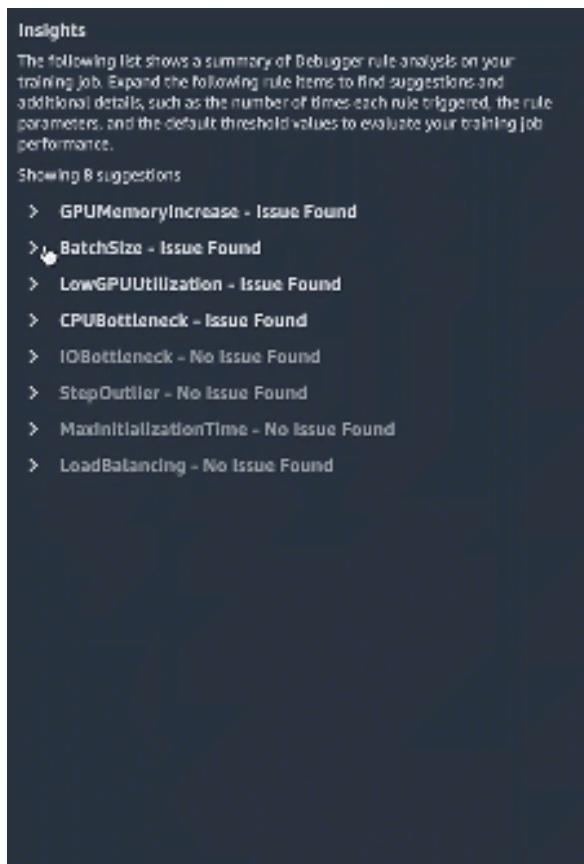
The **Resource intensive operations** section provides more detailed profiling results that show what operations of the training job were compute intensive. In the following example, it shows that the convolutional neural network backward pass operators were the most resource intensive on the GPUs.

Top operations on GPU		
Percentage (%)	Cumulative time	CPU operator
14.51	13805340	CudnnConvolutionBackward
34.48	13885079	subtot_convolution_backward
8.15	24028970	conv2d
6.11	24623438	convolution
6.08	2450894	_convolution
6.09	2421585	cuDNN_convolution
3.55	1351617	CudnnBatchNormBackward
3.31	1332746	custom_batch_norm_backward

## Insights

In the **Insights** pane, you can find training issues detected by Debugger built-in rules. You can expand each of the list to find useful insights, suggestions, a description of the rule, and criteria of triggering the rule.

For more information about the Debugger built-in rules, see [List of Debugger Built-in Rules \(p. 1626\)](#).

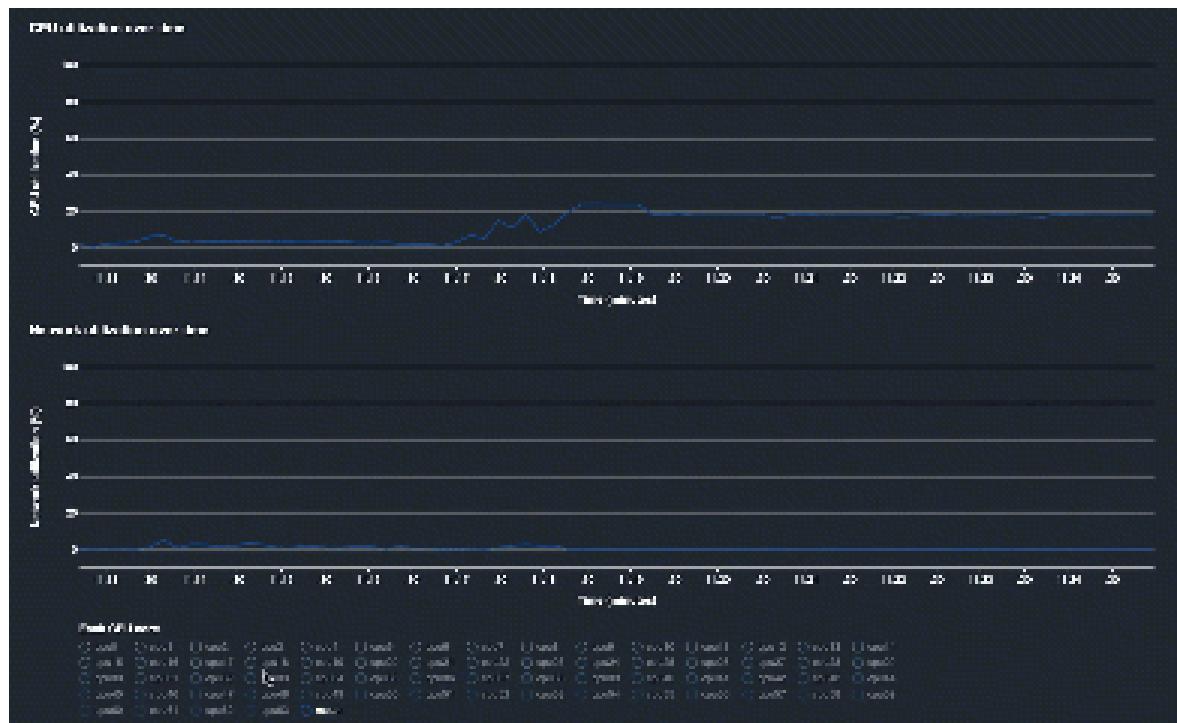


## Debugger Insights – Nodes

On the Studio Debugger Insight **Nodes** tab, Debugger provides detailed graphs that track each compute node on which your training jobs are running.

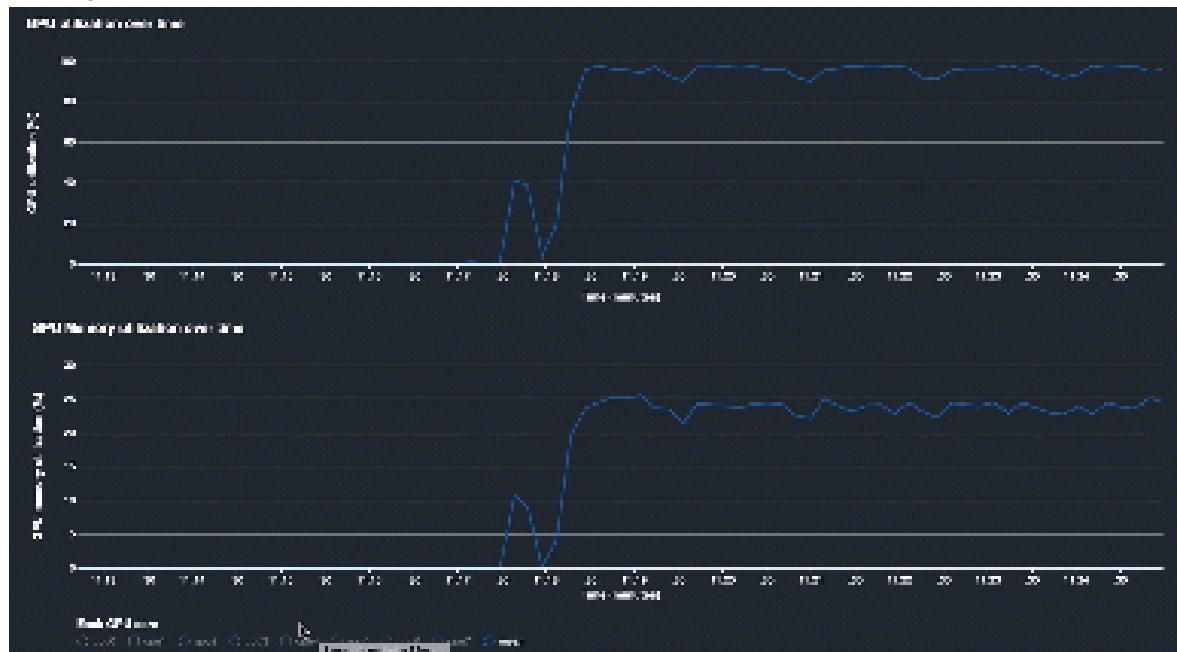
### CPU and Network utilization

The first two graphs show CPU utilization and network utilization over time. By default, the graphs show the mean values: the average of CPU and network utilization over the total number of CPU cores. You can select one or more CPU cores, by selecting on the labels, to graph them on single chart and compare utilization across cores. The timeline graphs are interactive, and the two graphs are synced up. You can drag and zoom in and out to see specific time windows that you want to have a closer look.



## GPU and GPU memory utilization

The following graphs show GPU utilization and GPU memory utilization over time. By default, the graphs show the mean utilization rate over time. You can select the GPU core labels to see the utilization rate of each core. Taking the mean of utilization rate over the total number of GPU cores shows the mean utilization of the entire hardware system resource. By looking at the mean utilization rate, you can check the overall system resource usage of EC2 instance. The following figure shows an example training job on an ml.p3.16xlarge instance with 8 GPU cores. You can monitor if the training job is well distributed, fully utilizing all GPUs.



### Overall system utilization over time

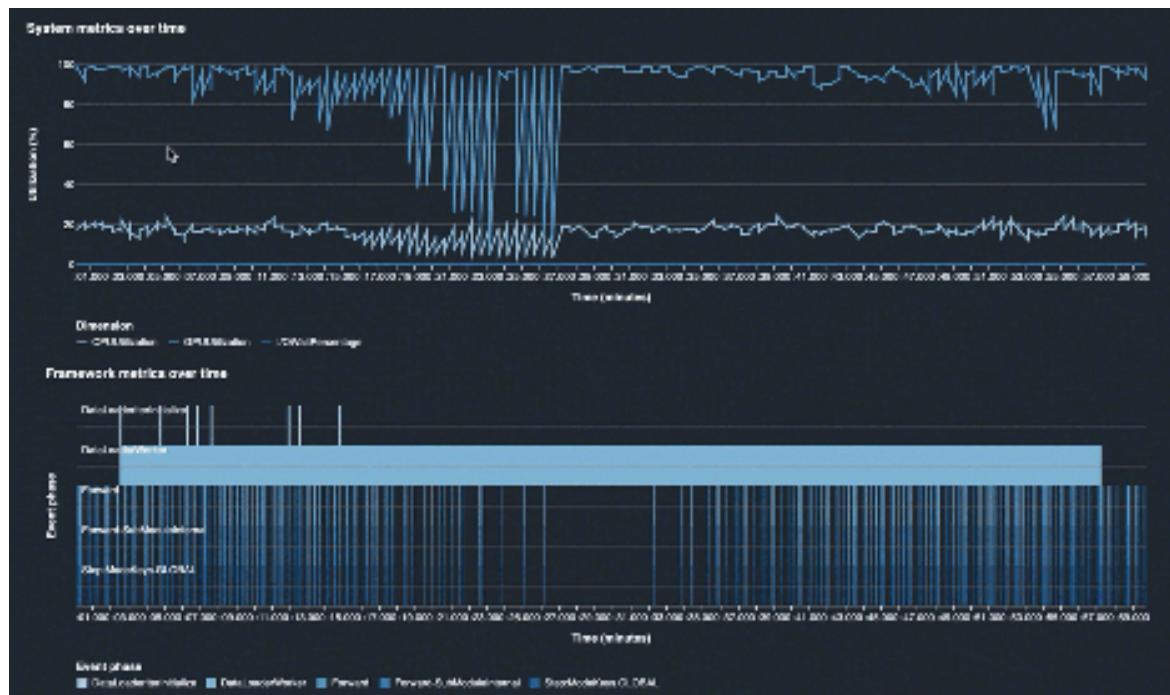
The following heatmap shows the entire system utilization over time projected onto the two-dimensional plot. Every CPU and GPU cores are listed in the vertical axis, and the utilization is recorded over time with colors. See the labeled colorbar on the right side of the plot to find out what color level corresponds to what utilization rate. For example, in the following heatmap, after initialization phase has ended around Sun 23:18, you can find that the training job fully utilizes an ml.p3.16xlarge instance; the GPU cores are fully utilized and the CPUs are moderately used for processing Python operations. There were several CPU bottleneck problems scattered across the CPUs at different times.



### System resource utilization over time and framework event phase

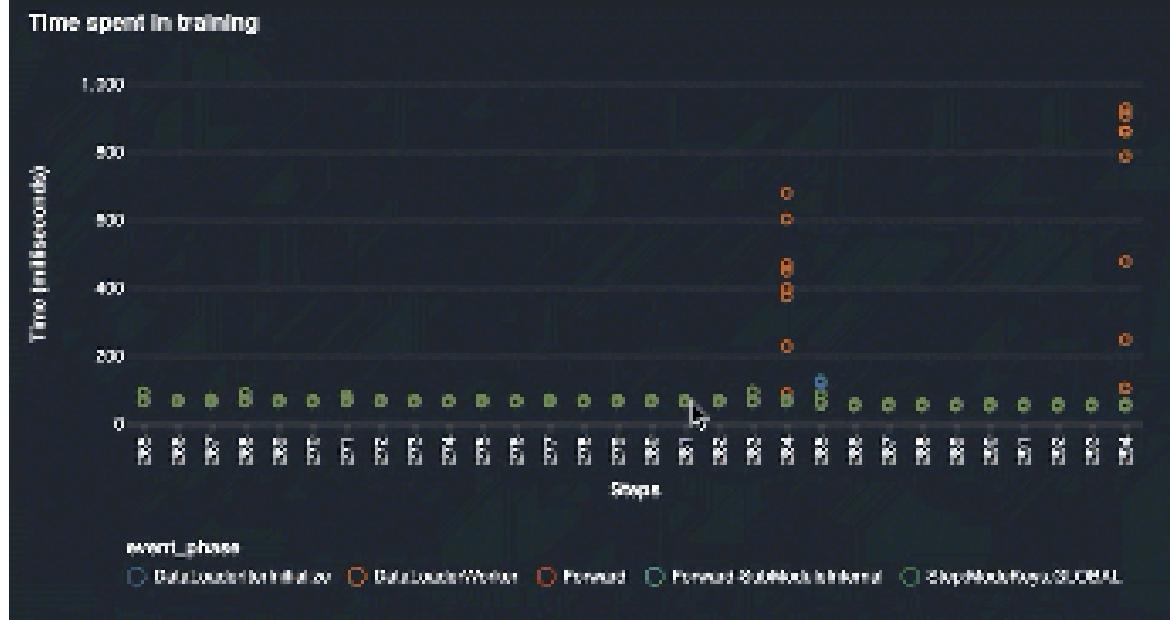
The **System metrics over time** graph shows the overall CPU, GPU, and data I/O utilization. The **Framework metrics over time** graph shows the framework metrics, which are the framework event phases that you can correlate with the **System metrics over time** graph.

You can select a time interval of interest in the system resource usage timeline, and the framework event phase spots the interval to show what events happened during the selected time interval. In each event phase block, you can find what time interval was actually spent for training loop, and break the training loop into backward pass and forward pass events. Overall, you can see that actual training time intervals are occupying only a small percentage over the entire training time.



### Time spent in training

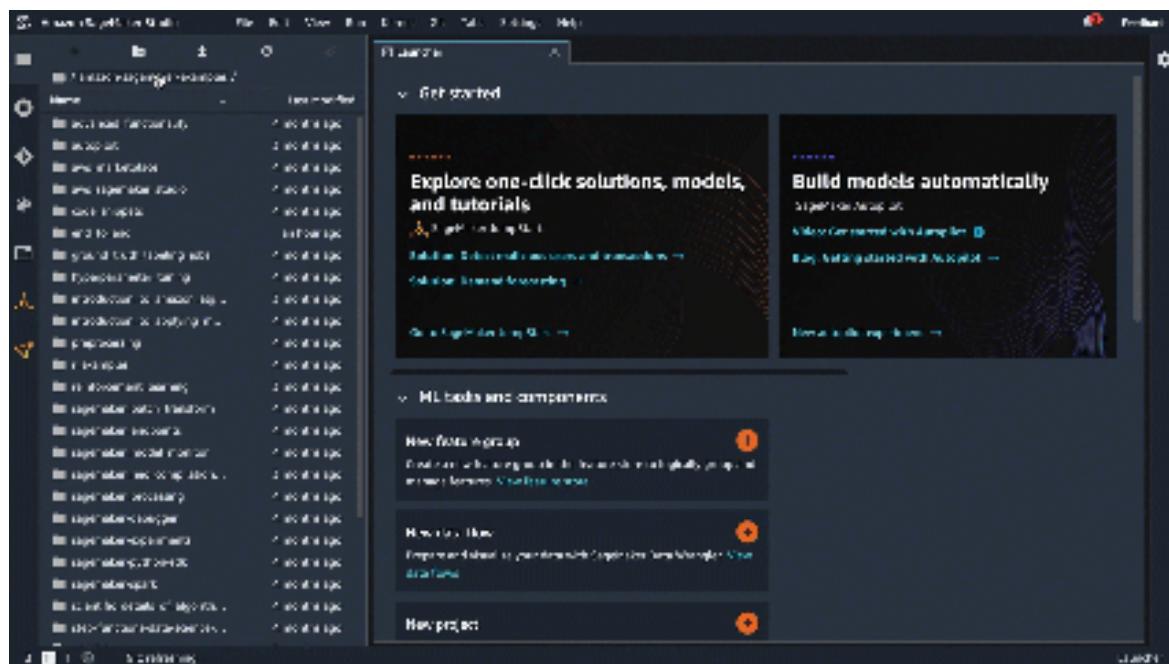
In the following graph, framework metrics from the last 30 steps of the last training loop are captured. This graph shows cumulative time spent by different events in each step.



## Shut Down the SageMaker Debugger Insights Instance

When you are not using the Debugger insights dashboard, it is important to shut down the instance on which it runs to avoid incurring additional fees.

### To shut down the Debugger insights instance in Studio



1. In Studio, select the **Running Instances and Kernels** icon ( ).
2. Under the **RUNNING APPS** list, look for the **sagemaker-debugger-1.0** app. Select the shut down icon ( ) next to the app. The Debugger insights dashboards run on an `m1.m5.4xlarge` instance. This instance also disappears from the **RUNNING INSTANCES** when you shut down the **sagemaker-debugger-1.0** app.

## SageMaker Debugger on Studio Experiments

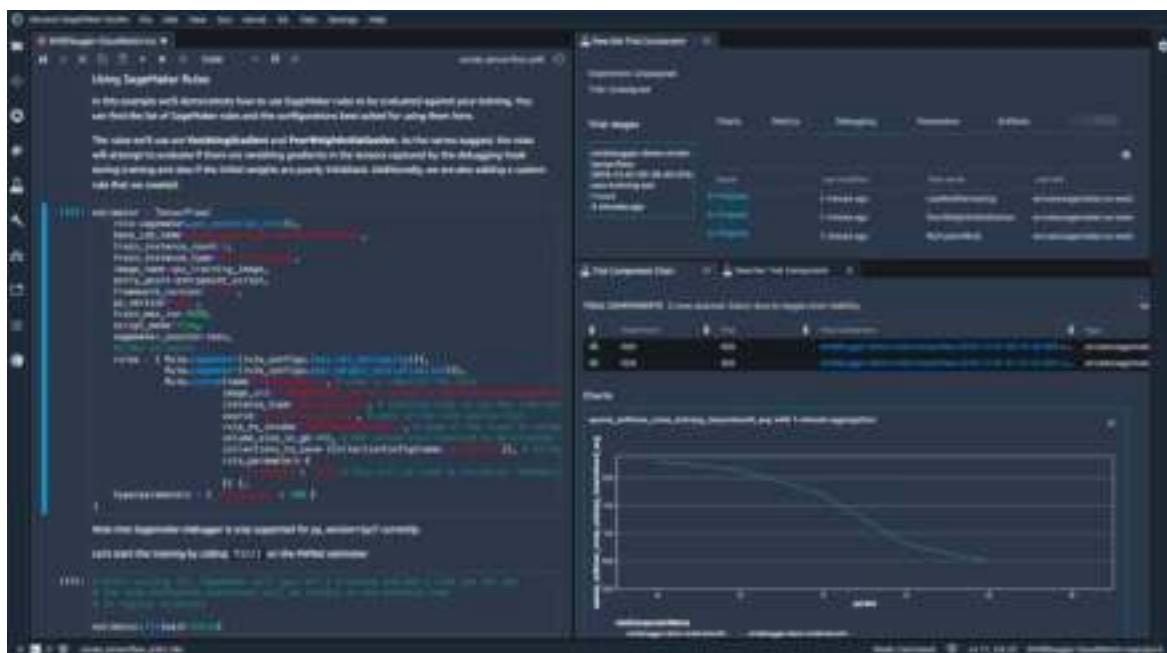
In this section, you learn how to use the Debugger in Studio Experiments. You can select any training jobs from the Experiment trial list to see the model output data graphs, such as accuracy and loss curves, debugging built-in rule status, and Debugger configuration information for debugging.

### Visualize Tensors Using SageMaker Debugger and Studio

SageMaker Studio provides visualizations to interpret tensor outputs that are captured by Debugger.

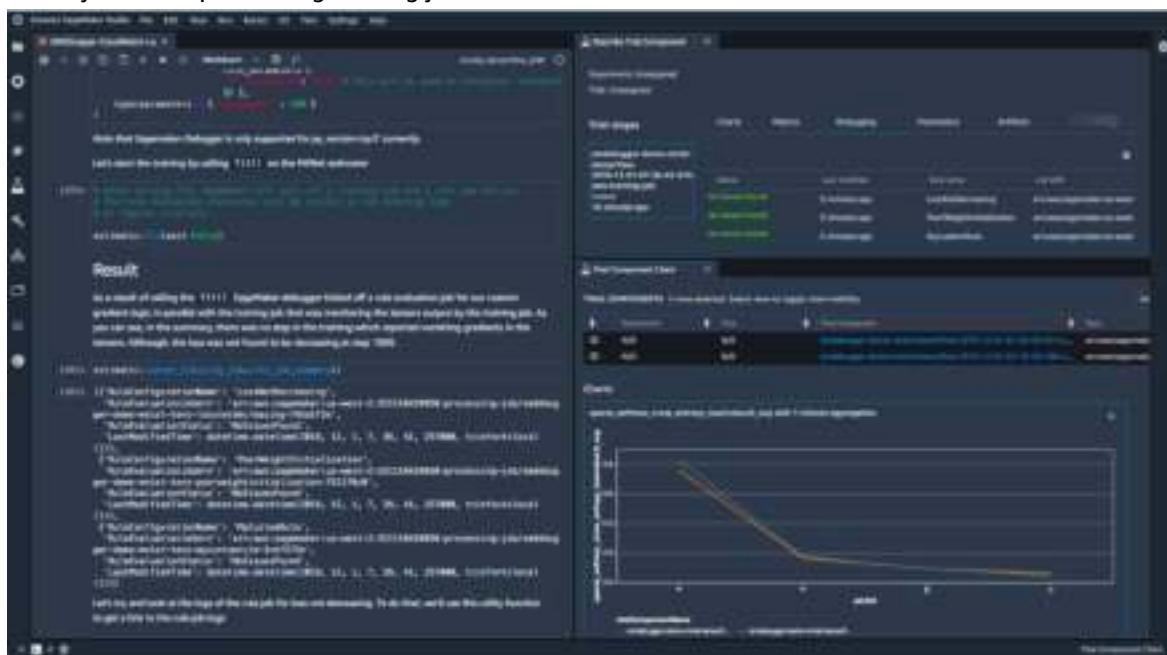
#### Loss Curves While Training Is in Progress

The following screenshot shows visualizations of loss curves for training. The training is in progress.



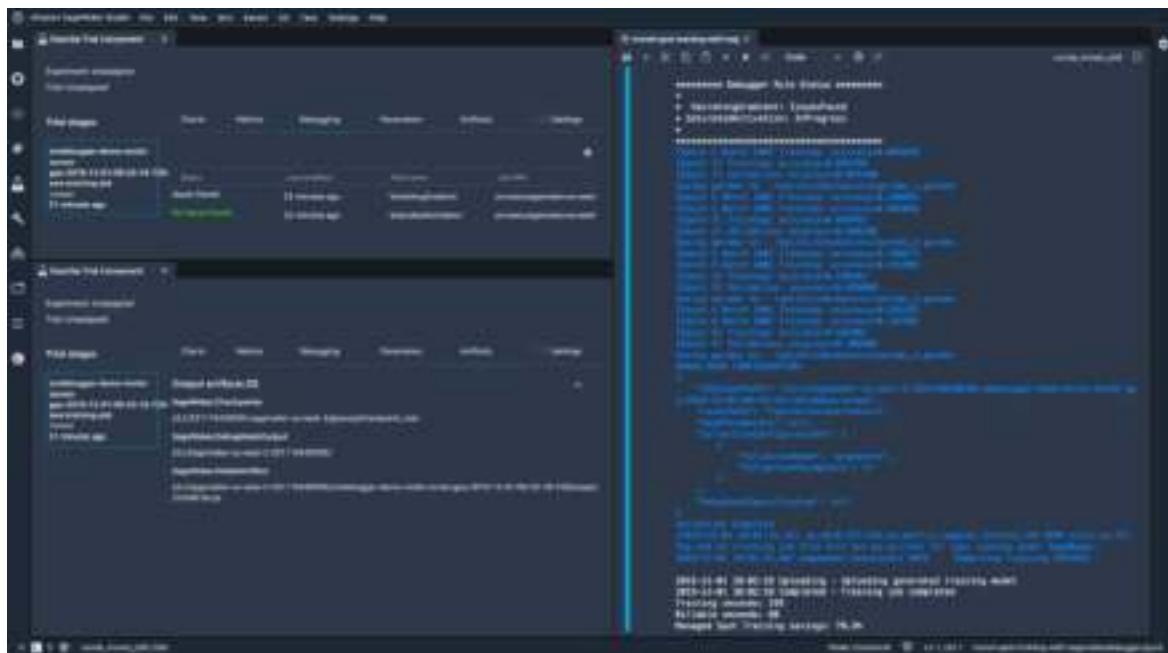
## Analyzing Training Jobs: Comparing Loss Curves Across Multiple Jobs

SageMaker Studio enables you to compare across multiple jobs (in this case, the loss). This helps you identify the best-performing training jobs.



## Rules Triggering and Logs from Jobs

When rules are triggered for anomalous conditions, SageMaker Studio presents logs for the failing rule. This enables you to analyze the causes of the condition.



## SageMaker Debugger Interactive Reports

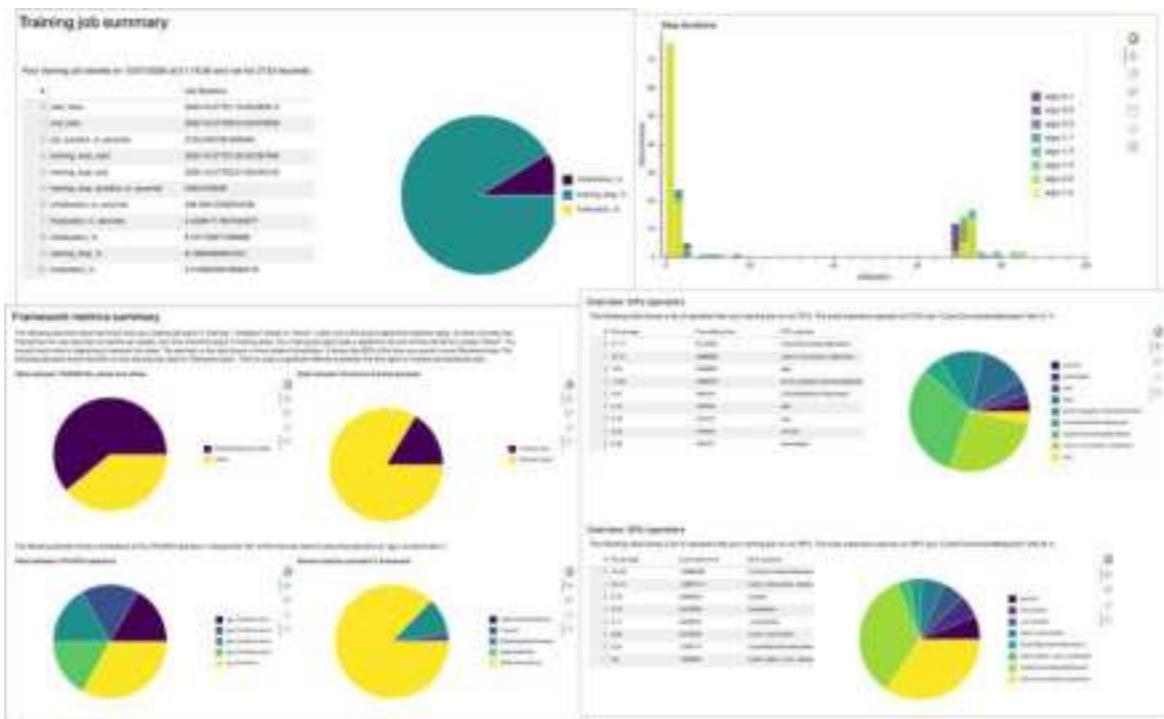
Receive training and profiling reports autogenerated by Debugger. The Debugger reports provide insights into your training jobs and suggest recommendations to improve your model performance. The following screenshot shows a collage of the Debugger profiling report. To learn more, see [SageMaker Debugger Profiling Report \(p. 1701\)](#).

### Note

You can download a Debugger reports while your training job is running or after the job has finished. During training, Debugger concurrently updates the report reflecting the current rules' evaluation status. You can download a complete Debugger report only after the training job has completed.

### Topics

- [SageMaker Debugger Profiling Report \(p. 1701\)](#)
- [SageMaker Debugger XGBoost Training Report \(p. 1711\)](#)



## SageMaker Debugger Profiling Report

For any SageMaker training jobs, the Debugger [ProfilerReport \(p. 1628\)](#) rule invokes all of the [monitoring and profiling rules \(p. 1626\)](#) and aggregates the rule analysis into a comprehensive report. Following this guide, download the report using the [Amazon SageMaker Python SDK](#) or the S3 console, and learn what you can interpret from the profiling results.

### Topics

- [Download a Debugger Profiling Report \(p. 1701\)](#)
- [Debugger Profiling Report Walkthrough \(p. 1704\)](#)

## Download a Debugger Profiling Report

Download the Debugger profiling report while your training job is running or after the job has finished using the [Amazon SageMaker Python SDK](#) and Amazon Command Line Interface (CLI).

### Tip

You can also download the report with one click and no additional scripting through the SageMaker Studio Debugger insights dashboard. To find out how to download the report from Studio, see [Open Amazon SageMaker Debugger Insights Dashboard \(p. 1686\)](#).

Download using SageMaker Python SDK and Amazon CLI

1. Check the current job's default S3 output base URI.

```
estimator.output_path
```

2. Check the current job name.

```
estimator.latest_training_job.job_name
```

- The Debugger profiling report is stored under <default-s3-output-base-uri>/<training-job-name>/rule-output. Configure the rule output path as follows:

```
rule_output_path = estimator.output_path + estimator.latest_training_job.job_name +
    "/rule-output"
```

- To check if the report is generated, list directories and files recursively under the rule\_output\_path using aws s3 ls with the --recursive option.

```
! aws s3 ls {rule_output_path} --recursive
```

This should return a complete list of files under an autogenerated folder that's named as ProfilerReport-1234567890. The folder name is a combination of strings: ProfilerReport and a unique 10-digit tag based on the Unix timestamp when the ProfilerReport rule is initiated.



The profiler-report.html is an autogenerated profiling report by Debugger. The remaining files are the built-in rule analysis components stored in JSON and a Jupyter notebook that are used to aggregate them into the report.

- Download the files recursively using aws s3 cp. The following command saves all of the rule output files to the ProfilerReport-1234567890 folder under the current working directory.

```
! aws s3 cp {rule_output_path} ./ --recursive
```

#### Tip

If using a Jupyter notebook server, run !pwd to double check the current working directory.

- Under the /ProfilerReport-1234567890/profiler-output directory, open profiler-report.html. If using JupyterLab, choose Trust HTML to see the autogenerated Debugger profiling report.



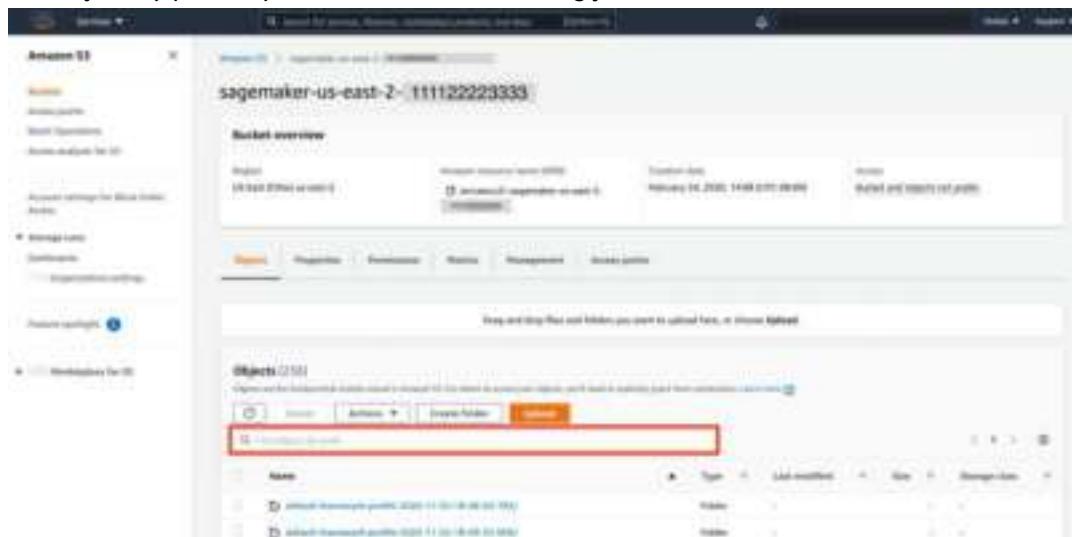
- Open the profiler-report.ipynb file to explore how the report is generated. You can also customize and extend the profiling report using the Jupyter notebook file.

#### Download using Amazon S3 Console

- Sign in to the Amazon Web Services Management Console and open the Amazon S3 console at <https://console.amazonaws.cn/s3/>.
- Search for the base S3 bucket. For example, if you haven't specified any base job name, the base S3 bucket name should be in the following format: sagemaker-<region>-111122223333. Look up the base S3 bucket through the *Find bucket by name* field.



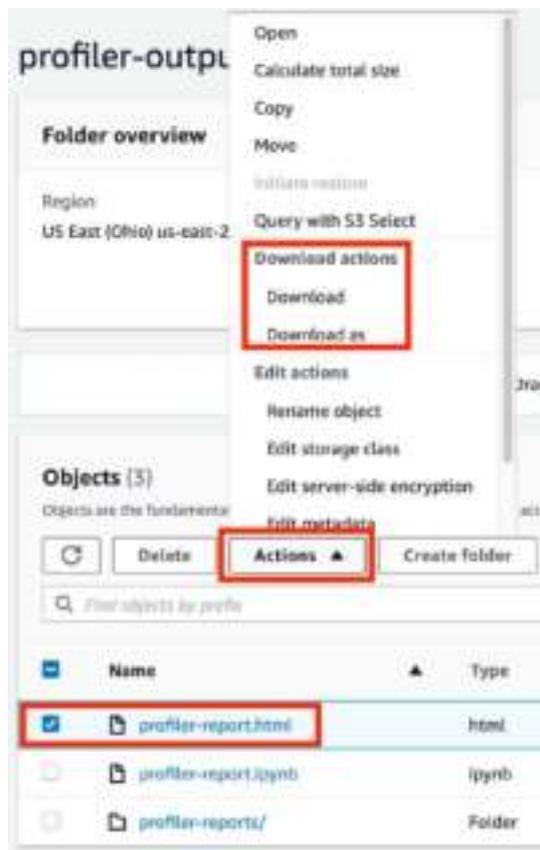
- In the base S3 bucket, look up the training job name by specifying your job name prefix into the *Find objects by prefix* input field. Choose the training job name.



- In the training job's S3 bucket, there must be three subfolders for training data collected by Debugger: **debug-output/**, **profiler-output/**, and **rule-output/**. Choose **rule-output/**.



- In the **rule-output/** folder, choose **ProfilerReport-1234567890**, and choose **profiler-output/** folder. The **profiler-output/** folder contains **profiler-report.html** (the autogenerated profiling report in html), **profiler-report.ipynb** (a Jupyter notebook with scripts that are used for generating the report), and a **profiler-report/** folder (contains rule analysis JSON files that are used as components of the report).
- Select the **profiler-report.html** file, choose **Actions**, and **Download**.



7. Open the downloaded **profiler-report.html** file in a web browser.

#### Note

If you started your training job without configuring the Debugger-specific parameters, Debugger generates the report based only on the system monitoring rules because the Debugger parameters are not configured to save framework metrics. To enable framework metrics profiling and receive an extended Debugger profiling report, configure the `profiler_config` parameter when constructing or updating SageMaker estimators.

To learn how to configure the `profiler_config` parameter before starting a training job, see [Configure Debugger Framework Profiling \(p. 1598\)](#).

To update the current training job and enable framework metrics profiling, see [Update Debugger Framework Profiling Configuration \(p. 1602\)](#).

## Debugger Profiling Report Walkthrough

This section walks you through the Debugger profiling report section by section. The profiling report is generated based on the built-in rules for monitoring and profiling. The report shows result plots only for the rules that found issues.

### Topics

- [Training Job Summary \(p. 1705\)](#)
- [System Usage Statistics \(p. 1706\)](#)
- [Framework metrics summary \(p. 1706\)](#)
- [Rules Summary \(p. 1708\)](#)
- [Analyzing the Training Loop – Step Durations \(p. 1709\)](#)
- [GPU Utilization Analysis \(p. 1709\)](#)

- [Batch Size \(p. 1709\)](#)
- [CPU Bottlenecks \(p. 1710\)](#)
- [I/O Bottlenecks \(p. 1711\)](#)
- [LoadBalancing in Multi-GPU Training \(p. 1711\)](#)
- [GPU Memory Analysis \(p. 1711\)](#)

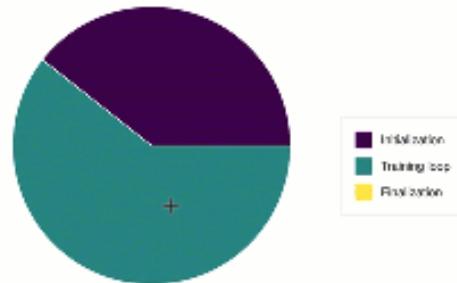
## Training Job Summary

At the beginning of the report, Debugger provides a summary of your training job. In this section, you can overview the time durations and timestamps at different training phases.

### Training job summary

The following table gives a summary about the training job. The table includes information about when the training job started and ended, how much time initialization, training loop and finalization took. Your training job started on 11/29/2020 at 23:12:42, and ran for 737 seconds.

#	Job Statistic
0	Start time
1	23:12:42 11/29/2020
2	End time
3	23:54:59 11/29/2020
4	Job duration
5	737 seconds
6	Training loop start
7	23:17:31 11/29/2020
8	Training loop end
9	23:24:59 11/29/2020
10	Training loop duration
11	115 seconds
12	Initialization time
13	285 seconds
14	Finalization time
15	0 seconds
16	Initialization
17	39 %
18	Training loop
19	60 %
20	Finalization
21	0 %



The summary table contains the following information:

- **start\_time** – The exact time when the training job started.
- **end\_time** – The exact time when the training job finished.
- **job\_duration\_in\_seconds** – The total training time from the **start\_time** to the **end\_time**.
- **training\_loop\_start** – The exact time when the first step of the first epoch has started.
- **training\_loop\_end** – The exact time when the last step of the last epoch has finished.
- **training\_loop\_duration\_in\_seconds** – The total time between the training loop start time and the training loop end time.
- **initialization\_in\_seconds** – Time spent on initializing the training job. The initialization phase covers the period from the **start\_time** to the **training\_loop\_start** time. The initialization time is spent on compiling the training script, starting the training script, creating and initializing the model, initiating EC2 instances, and downloading training data.
- **finalization\_in\_seconds** – Time spent on finalizing the training job, such as finishing the model training, updating the model artifacts, and closing the EC2 instances. The finalization phase covers the period from the **training\_loop\_end** time to the **end\_time**.
- **initialization (%)** – The percentage of time spent on **initialization** over the total **job\_duration\_in\_seconds**.
- **training loop (%)** – The percentage of time spent on **training loop** over the total **job\_duration\_in\_seconds**.
- **finalization (%)** – The percentage of time spent on **finalization** over the total **job\_duration\_in\_seconds**.

## System Usage Statistics

In this section, you can see an overview of system utilization statistics.

### System usage statistics

The 95th percentile of the total GPU utilization on node algo-2 is 74%. GPUs on node algo-2 are well utilized.

The following table shows usage statistics per worker node such as total CPU and GPU utilization, total CPU and memory footprint. The table also includes total I/O wait time and total sent/received bytes. The table shows min and max values as well as p99, p90 and p50 percentiles.

#	node -	metric	unit	max	p99	p90	p50	min
0	algo-1	Network	bytes	218817581.57	188.02	0	0	0
1	algo-1	I/O	percentage	13.2669129	8.862831250000000	0.1955937499999999	0	0
2	algo-1	CPU memory	percentage	30.25	26.25	21	0	0
3	algo-1	GPU	percentage	78	74.8	74.25	0	0
4	algo-1	CPU memory	percentage	8.06	6.04	4.98	2.17	0.33
5	algo-1	CPU	percentage	32.068828	22.62913125000000	17.034	3.703420000000000	0
6	algo-2	Network	bytes	4135.24	0	0	0	0
7	algo-2	I/O	percentage	29.1675	8.158290000000000	1.747613499999999	0	0
8	algo-2	CPU memory	percentage	38	31.75	21.75	0	0
9	algo-2	GPU	percentage	78	74.8	74.25	0	0
10	algo-2	CPU memory	percentage	8.06	6.04	4.99	2.17	0.39
11	algo-2	CPU	percentage	36.04370499999999	25.89998710000000	18.034298675	3.77829125	0

The Debugger profiling report includes the following information:

- **node** – Lists the name of nodes. If using distributed training on multiple nodes (multiple EC2 instances), the node names are in format of algo-n.
- **metric** – The system metrics collected by Debugger: CPU, GPU, CPU memory, GPU memory, I/O, and Network metrics.
- **unit** – The unit of the system metrics.
- **max** – The maximum value of each system metric.
- **p99** – The 99th percentile of each system utilization.
- **p90** – The 95th percentile of each system utilization.
- **p50** – The 50th percentile (median) of each system utilization.
- **min** – The minimum value of each system metric.

### Framework metrics summary

In this section, the following pie charts show the breakdown of framework operations on CPUs and GPUs.

### Framework metrics summary

This section provides an overview of the training job metrics. Within the type of training jobs, all the time recorded by the framework is either spent on the forward pass or backward pass. Detailed metrics should be used for further analysis. For training jobs specifically in a configuration of three DGX-2H GPU nodes, the following chart is reporting in seconds on the loops. The picture on the right shows a more detailed breakdown, showing that one of the time was spent in inverse Cholesky pass. The following sections show details about training and validation passes.

Ratio between TRAIN/EVAL phase and others



Ratio between forward and backward pass



The following figure shows the breakdown of CPU/GPU operators. It is worth 10% of the time recorded by the framework on Train/Eval loop.

Ratio between CPU/GPU operators



General metrics recorded in framework



Each of the pie charts analyzes the collected framework metrics in various aspects as follows:

- **Ratio between TRAIN/EVAL phase and others** – Shows the ratio between time durations spent on different training phases.
- **Ratio between forward and backward pass** – Shows the ratio between time durations spent on forward and backward pass in the training loop.
- **Ratio between CPU/GPU operators** – Shows the ratio between time spent on operators running on CPU or GPU, such as convolutional operators.
- **General metrics recorded in framework** – Shows the ratio between time spent on major framework metrics, such as data loading, forward and backward pass.

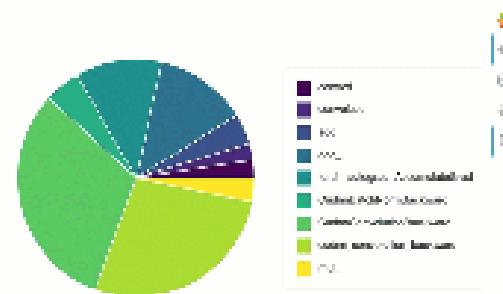
### Overview: CPU Operators

This section provides information of the CPU operators in detail. The table shows the percentage of the time and the absolute cumulative time spent on the most frequently called CPU operators.

## Overview: CPU operations

<sup>10</sup> The 1970 census showed 1,100,000 of the total 2000 population of 27,000,000 as being foreign-born.

#	Barcode ID	Current Status	CF Status
1	12345	On Hold	Queued for Review
2	56789	Completed	Completed
3	33333	... ...	...
4	44444	... ...	...
5	55555	... ...	...
6	66666	... ...	...
7	77777	... ...	...
8	88888	... ...	...
9	99999	... ...	...



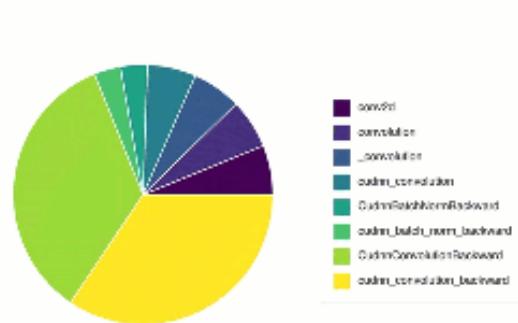
## Overview: GPU Operators

This section provides information of the GPU operators in detail. The table shows the percentage of the time and the absolute cumulative time spent on the most frequently called GPU operators.

## Overview: GPU operators

The following table shows a list of operators that your training job run on GPU. The most expensive operator on GPU was "CudnnConvolutionBackward" with 34 %.

#	Percentage	Cumulative time	GPU operator
0	34.48	189.9535	CustomConvolutionBatched
1	34.44	100.07210	custom_convolution_layer
2	6.18	94.9523	conv2d
3	5.13	24.73034	convolution
4	5.11	24.68905	convolution
5	5.06	24.44523	custom_convolution
6	3.84	19.87774	CustomBoxNormBackward
7	3.0	10.00003	custom_bwd_norm_layer



## Rules Summary

In this section, Debugger aggregates all of the rule evaluation results, analysis, rule descriptions, and suggestions.

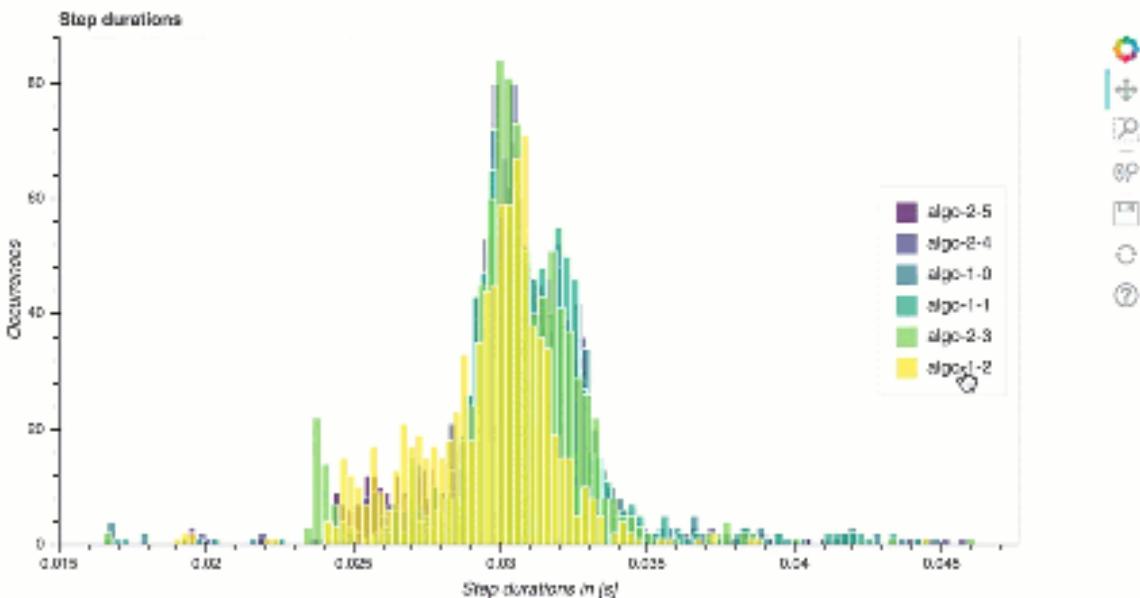
### **Ruler's Summary**

The following table lists the names of these 10 categories of the associated author names. The table is organized by the cause that triggered most frequently in your training data (the most frequent cause for each author). It has processed 5407 documents and suggested 503 times.

	Description	Implementation	Number of users who adopted	Number of users	Adopted percentage
TaskBoarding	Feature allows to automatically synchronize between multiple Gantt charts. Multitasking management can be increased as user can see and compare incoming tasks in parallel and prioritize them on priority basis. It also allows to quickly identify tasks which require urgent action without being affected by other tasks.	Create efficient multitasking strategy by effectively managing incoming tasks.	100	1000	100%
CostOptimization	Creates a Gantt chart to help you perform cost optimization. You can measure it based upon resources, team, budget and due date to identify inefficiencies or bottleneck activities.	Group all activities by resource planning code, change alternative planning strategy, increase batch size.	100	1000	100%
Reporting	Creates a Gantt chart to quickly review the duration and timing issues. To add it into the Gantt chart, the average Gantt currency (Resource, 25% and 40%) adjustment.	Run a quick analysis of your current Gantt chart.	100	1000	100%
Efficiency optimizer	A number which helps user to find major time wasting activities out of measured total time.	Create a report indicating tasks with lower efficiency. If it is not a resource limit or scope modification, fix it.	50	1000	50%
Efficiency optimizer	Create a Gantt chart to keep track of the using time. It helps to analyze a Gantt chart more deeply. It is used for better resource allocation.	Gantt chart helps to quickly identify overworking or idle workers.	100	1000	100%
Efficiency optimizer	A Gantt chart is a great tool for time saving (Gantt chart) to know, if they follow the Gantt chart. It is useful for tracking the progress of the project.	The Gantt chart is a visual representation of the project's tasks, their start and end dates, and dependencies.	50	1000	50%
Dashboard	Global portfolio risk detection. True for increased and enhanced quick decision making by using dashboards for learning. It shows real-time financial numbers to reveal indicators for more than 100 countries.	Create fast dashboard.	10	1000	1%
Dashboard	Creates a real time monitoring system to monitor your health care. The data comes from third party as well as own application in your own monitoring for the medical and 24/7 on dashboards to facilitate communication from different sources.	Dashboard for real time health.	5	1000	0.5%

## Analyzing the Training Loop – Step Durations

In this section, you can find a detailed statistics of step durations on each GPU core of each node. Debugger evaluates mean, maximum, p99, p95, p50, and minimum values of step durations, and evaluate step outliers. The following histogram shows the step durations captured on different worker nodes and GPUs. You can enable or disable the histogram of each worker by choosing the legends on the right side. You can check if there is a particular GPU that's causing step duration outliers.



## GPU Utilization Analysis

This section shows the detailed statistics about GPU core utilization based on LowGPUUtilization rule. It also summarizes the GPU utilization statistics, mean, p95, and p5 to determine if the training job is underutilizing GPUs.

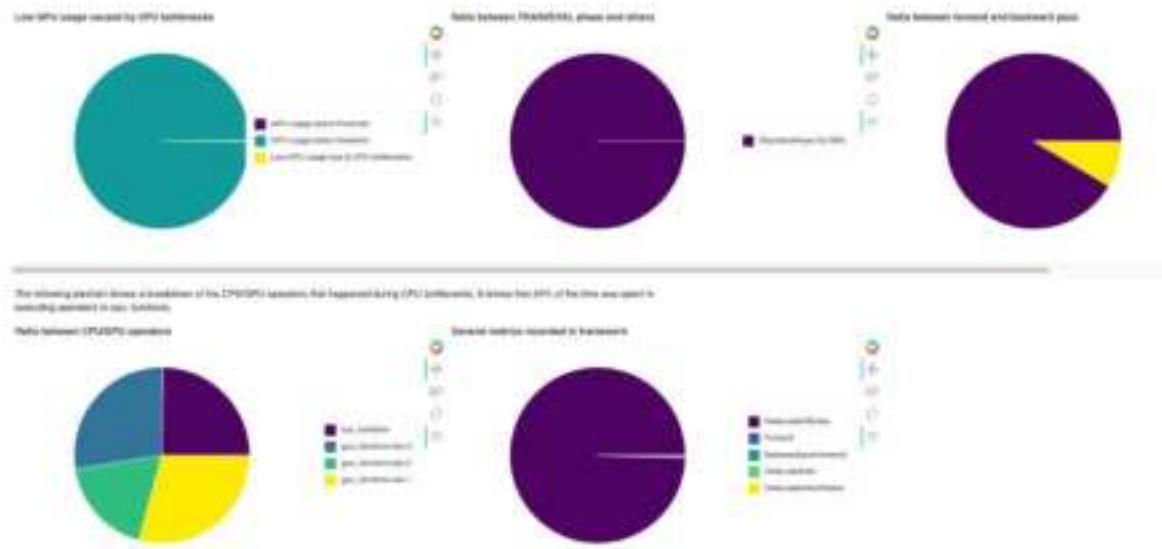
### Batch Size

This section shows the detailed statistics of total CPU utilization, individual GPU utilizations, and GPU memory footprints. The BatchSize rule determines if you need to change the batch size to better utilize the GPUs. You can check whether the batch size is too small resulting in underutilization or too large causing overutilization and out of memory issues. In the plot, the boxes show the p25 and p75 percentile ranges (filled with dark purple and bright yellow respectively) from the median (p50), and the error bars show the 5th percentile for the lower bound and 95th percentile for the upper bound.



## CPU Bottlenecks

In this section, you can drill down into the CPU bottlenecks that the CPUBottleneck rule detected from your training job. The rule checks if the CPU utilization is above `cpu_threshold` (90% by default) and also if the GPU utilization is below `gpu_threshold` (10% by default).



The pie charts show the following information:

- Low GPU usage caused by CPU bottlenecks** – Shows the ratio of data points between the ones with GPU utilization above and below the threshold and the ones that matches the CPU bottleneck criteria.
- Ratio between TRAIN/EVAL phase and others** – Shows the ratio between time durations spent on different training phases.
- Ratio between forward and backward pass** – Shows the ratio between time durations spent on forward and backward pass in the training loop.

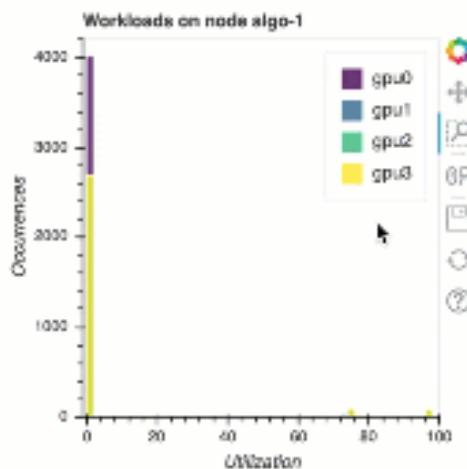
- **Ratio between CPU/GPU operators** – Shows the ratio between time durations spent on GPUs and CPUs by Python operators, such as data loader processes and forward and backward pass operators.
- **General metrics recorded in framework** – Shows major framework metrics and the ratio between time durations spent on the metrics.

## I/O Bottlenecks

In this section, you can find a summary of IO bottlenecks.

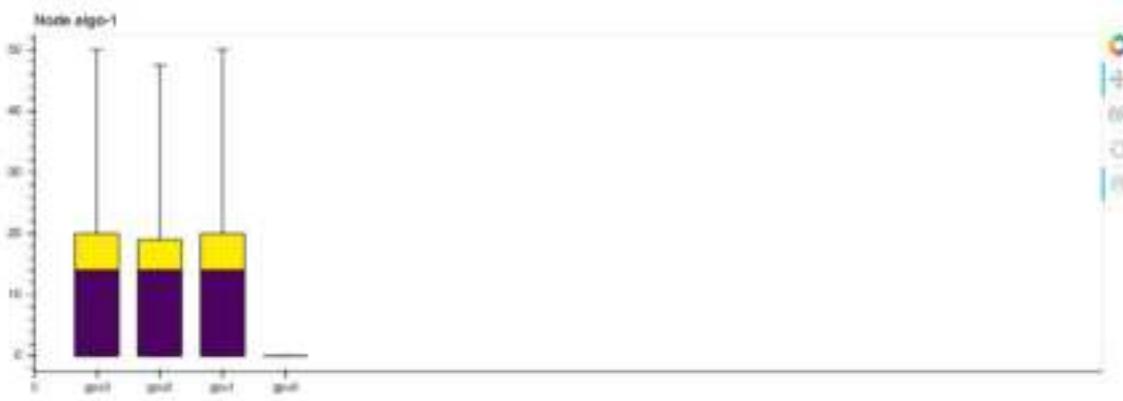
### LoadBalancing in Multi-GPU Training

In this section, you can identify workload balancing issue across GPUs.



## GPU Memory Analysis

In this section, you can analyze the GPU memory utilization collected by the GPUMemoryIncrease rule. In the plot, the boxes show the p25 and p75 percentile ranges (filled with dark purple and bright yellow respectively) from the median (p50), and the error bars show the 5th percentile for the lower bound and 95th percentile for the upper bound.



## SageMaker Debugger XGBoost Training Report

For SageMaker XGBoost training jobs, use the Debugger [CreateXgboostReport \(p. 1638\)](#) rule to receive a comprehensive training report of the training progress and results. Following this guide, specify the [CreateXgboostReport \(p. 1638\)](#) rule while constructing an XGBoost estimator, download the report using

the [Amazon SageMaker Python SDK](#) or the Amazon S3 console, and then you can interpret the profiling results.

### Topics

- [Construct a SageMaker XGBoost Estimator with the Debugger XGBoost Report Rule \(p. 1712\)](#)
- [Download the Debugger XGBoost Training Report \(p. 1713\)](#)
- [Debugger XGBoost Training Report Walkthrough \(p. 1715\)](#)

## Construct a SageMaker XGBoost Estimator with the Debugger XGBoost Report Rule

When you construct a SageMaker estimator for an XGBoost training job, specify the rule as shown in the following example code.

The [CreateXgboostReport \(p. 1638\)](#) rule collects the following output tensors from your training job:

- `hyperparameters` – Saves at the first step.
- `metrics` – Saves loss and accuracy every 5 steps.
- `feature_importance` – Saves every 5 steps.
- `predictions` – Saves every 5 steps.
- `labels` – Saves every 5 steps.

The output tensors are saved at a default S3 bucket. For example, `s3://sagemaker-<region>-<12digit_account_id>/<base-job-name>/debug-output/`.

Using the SageMaker generic estimator

```
import boto3
import sagemaker
from sagemaker.estimator import Estimator
from sagemaker import image_uris
from sagemaker.debugger import Rule, rule_configs

rules=[
    Rule.sagemaker(rule_configs.create_xgboost_report())
]

region = boto3.Session().region_name
xgboost_container=sagemaker.image_uris.retrieve("xgboost", region, "1.2-1")

estimator=Estimator(
    role=sagemaker.get_execution_role()
    image_uri=xgboost_container,
    base_job_name="debugger-xgboost-report-demo",
    instance_count=1,
    instance_type="ml.m5.2xlarge",

    # Add the Debugger XGBoost report rule
    rules=rules
)

estimator.fit(wait=False)
```

## Download the Debugger XGBoost Training Report

Download the Debugger XGBoost training report while your training job is running or after the job has finished using the [Amazon SageMaker Python SDK](#) and Amazon Command Line Interface (CLI).

Download using the SageMaker Python SDK and Amazon CLI

1. Check the current job's default S3 output base URI.

```
estimator.output_path
```

2. Check the current job name.

```
estimator.latest_training_job.job_name
```

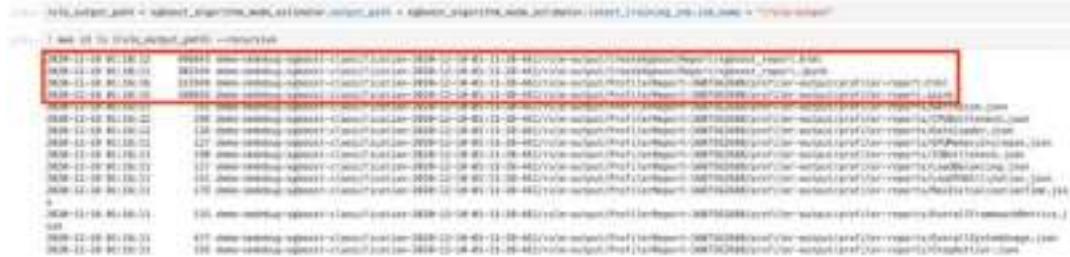
3. The Debugger XGBoost report is stored under <default-s3-output-base-uri>/<training-job-name>/rule-output. Configure the rule output path as follows:

```
rule_output_path = estimator.output_path + estimator.latest_training_job.job_name +  
    "/rule-output"
```

4. To check if the report is generated, list directories and files recursively under the rule\_output\_path using aws s3 ls with the --recursive option.

```
! aws s3 ls {rule_output_path} --recursive
```

This should return a complete list of files under autogenerated folders that are named CreateXgboostReport and ProfilerReport-1234567890. The XGBoost training report is stored in the CreateXgboostReport, and the profiling report is stored in the ProfilerReport-1234567890 folder. To learn more about the profiling report generated by default with the XGBoost training job, see [SageMaker Debugger Profiling Report \(p. 1701\)](#).



The xgboost\_report.html is an autogenerated XGBoost training report by Debugger. The xgboost\_report.ipynb is a Jupyter notebook that's used to aggregate training results into the report. You can download all of the files, browse the HTML report file, and modify the report using the notebook.

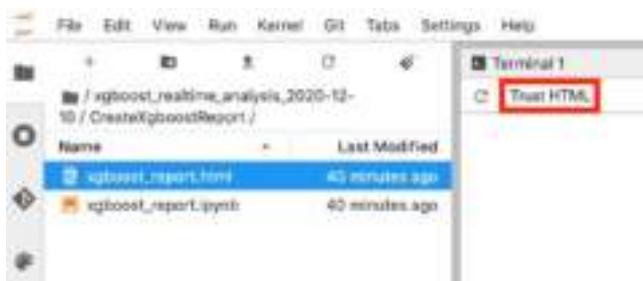
5. Download the files recursively using aws s3 cp. The following command saves all of the rule output files to the ProfilerReport-1234567890 folder under the current working directory.

```
! aws s3 cp {rule_output_path} ./ --recursive
```

### Tip

If you are using a Jupyter notebook server, run !pwd to verify the current working directory.

6. Under the /CreateXgboostReport directory, open xgboost\_report.html. If you are using JupyterLab, choose **Trust HTML** to see the autogenerated Debugger training report.



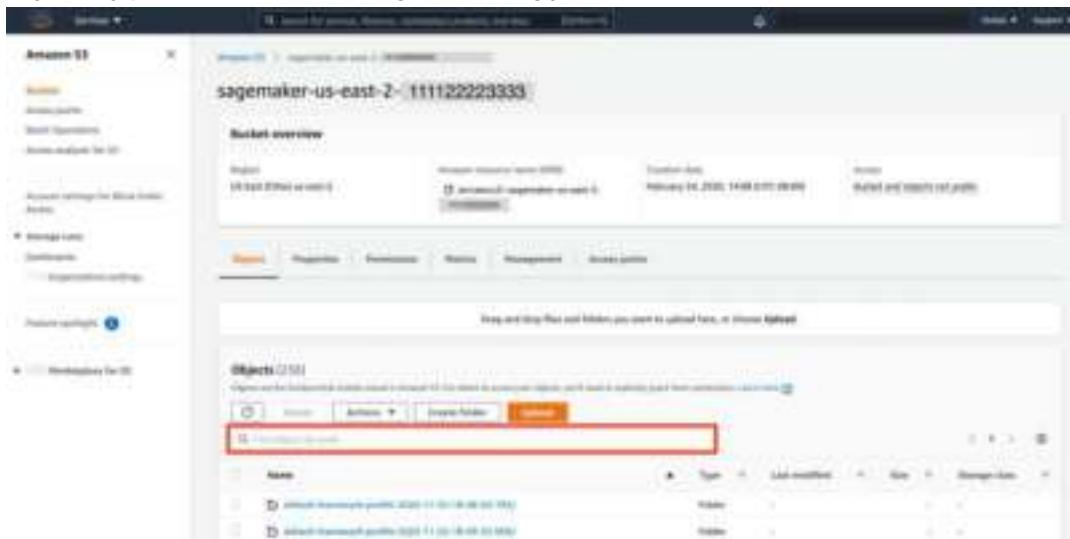
7. Open the `xgboost_report.ipynb` file to explore how the report is generated. You can customize and extend the training report using the Jupyter notebook file.

Download using the Amazon S3 console

1. Sign in to the Amazon Web Services Management Console and open the Amazon S3 console at <https://console.amazonaws.cn/s3/>.
2. Search for the base S3 bucket. For example, if you haven't specified any base job name, the base S3 bucket name should be in the following format: `sagemaker-<region>-111122223333`. Look up the base S3 bucket through the **Find bucket by name** field.



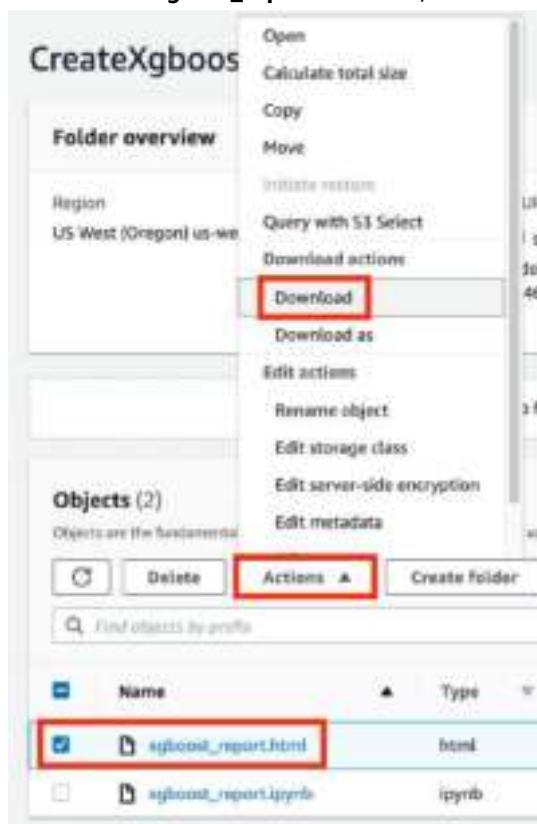
3. In the base S3 bucket, look up the training job name by entering your job name prefix in **Find objects by prefix** and then choosing the training job name.



4. In the training job's S3 bucket, choose **rule-output/** subfolder. There must be three subfolders for training data collected by Debugger: **debug-output/**, **profiler-output/**, and **rule-output/**.



5. In the **rule-output/** folder, choose the **CreateXgboostReport/** folder. The folder contains **xbgoost\_report.html** (the autogenerated report in html) and **xbgoost\_report.ipynb** (a Jupyter notebook with scripts that are used for generating the report).
6. Choose the **xbgoost\_report.html** file, choose **Download actions**, and then choose **Download**.



7. Open the downloaded **xbgoost\_report.html** file in a web browser.

## Debugger XGBoost Training Report Walkthrough

This section walks you through the Debugger XGBoost training report. The report is automatically aggregated depending on the output tensor regex, recognizing what type of your training job is among binary classification, multiclass classification, and regression.

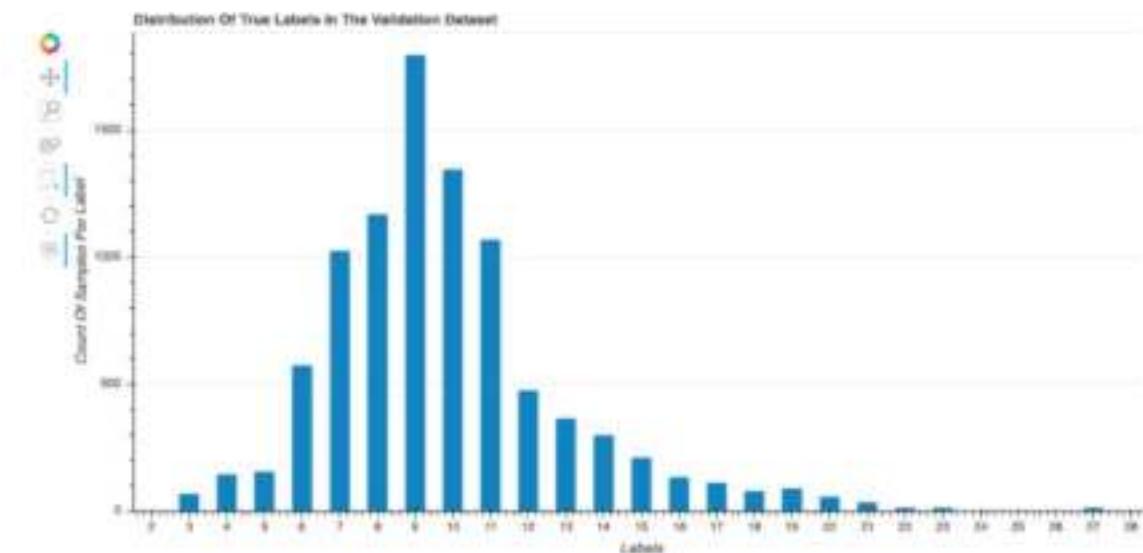
### Topics

- [Distribution of True Labels of the Dataset \(p. 1716\)](#)
- [Loss versus Step Graph \(p. 1716\)](#)

- [Feature Importance \(p. 1717\)](#)
- [Confusion Matrix \(p. 1718\)](#)
- [Evaluation of the Confusion Matrix \(p. 1719\)](#)
- [Accuracy Rate of Each Diagonal Element Over Iteration \(p. 1720\)](#)
- [Receiver Operating Characteristic Curve \(p. 1721\)](#)
- [Distribution of Residuals at the Last Saved Step \(p. 1722\)](#)
- [Absolute Validation Error per Label Bin Over Iteration \(p. 1723\)](#)

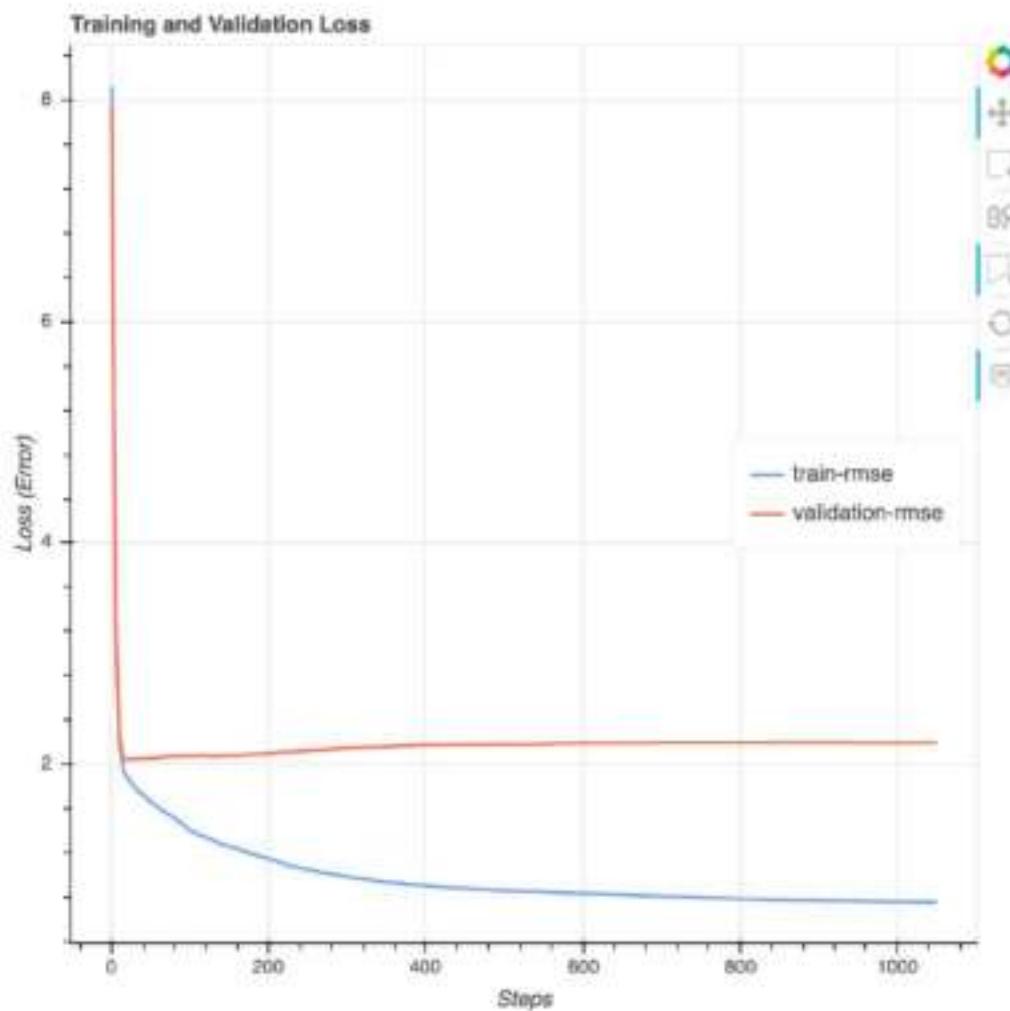
### Distribution of True Labels of the Dataset

This histogram shows the distribution of labeled classes (for classification) or values (for regression) in your original dataset. Skewness in your dataset could contribute to inaccuracies. This visualization is available for the following model types: binary classification, multiclassification, and regression.



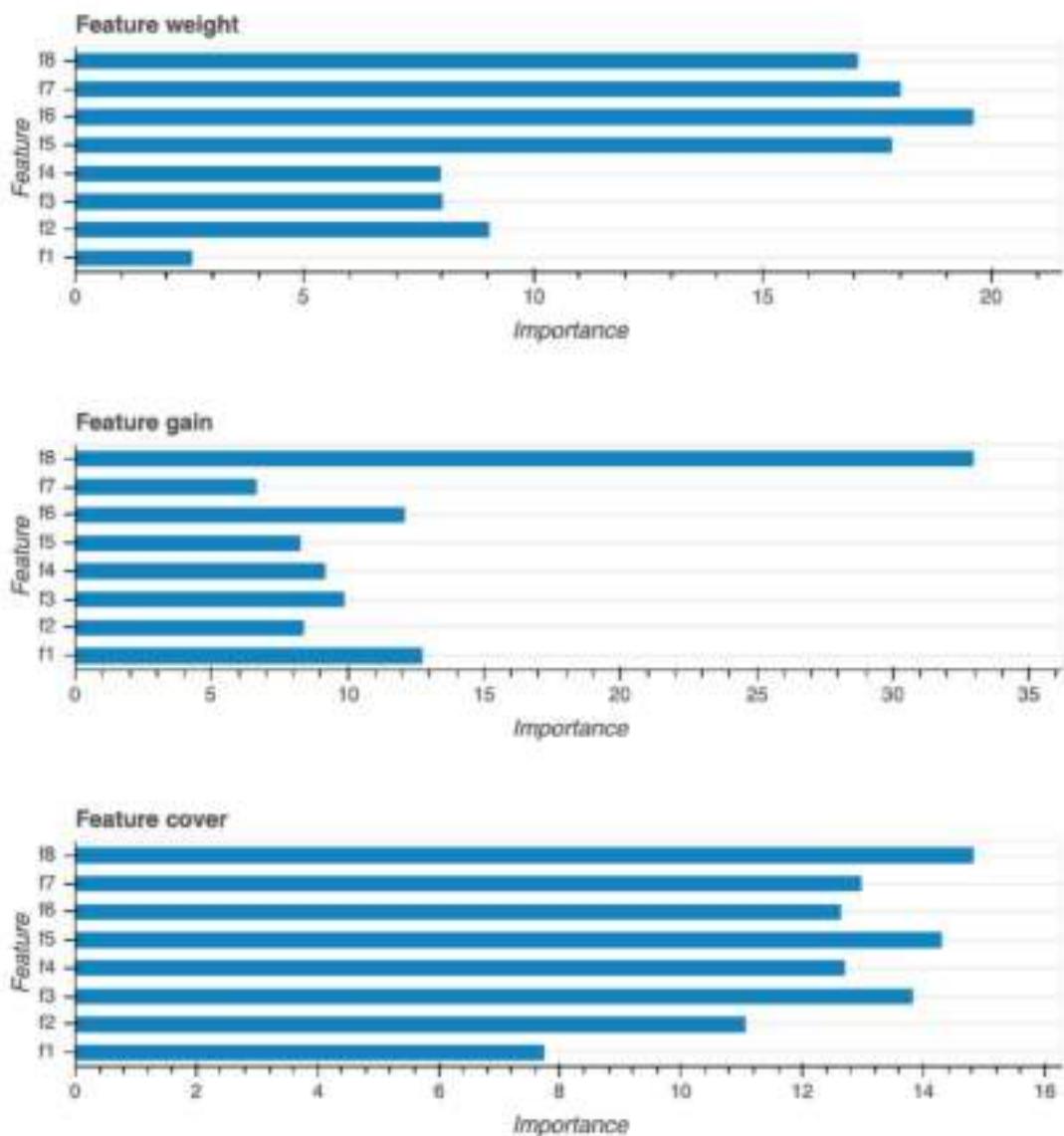
### Loss versus Step Graph

This is a line chart that shows the progression of loss on training data and validation data throughout training steps. The loss is what you defined in your objective function, such as mean squared error. You can gauge whether the model is overfit or underfit from this plot. This section also provides insights that you can use to determine how to resolve the overfit and underfit problems. This visualization is available for the following model types: binary classification, multiclassification, and regression.



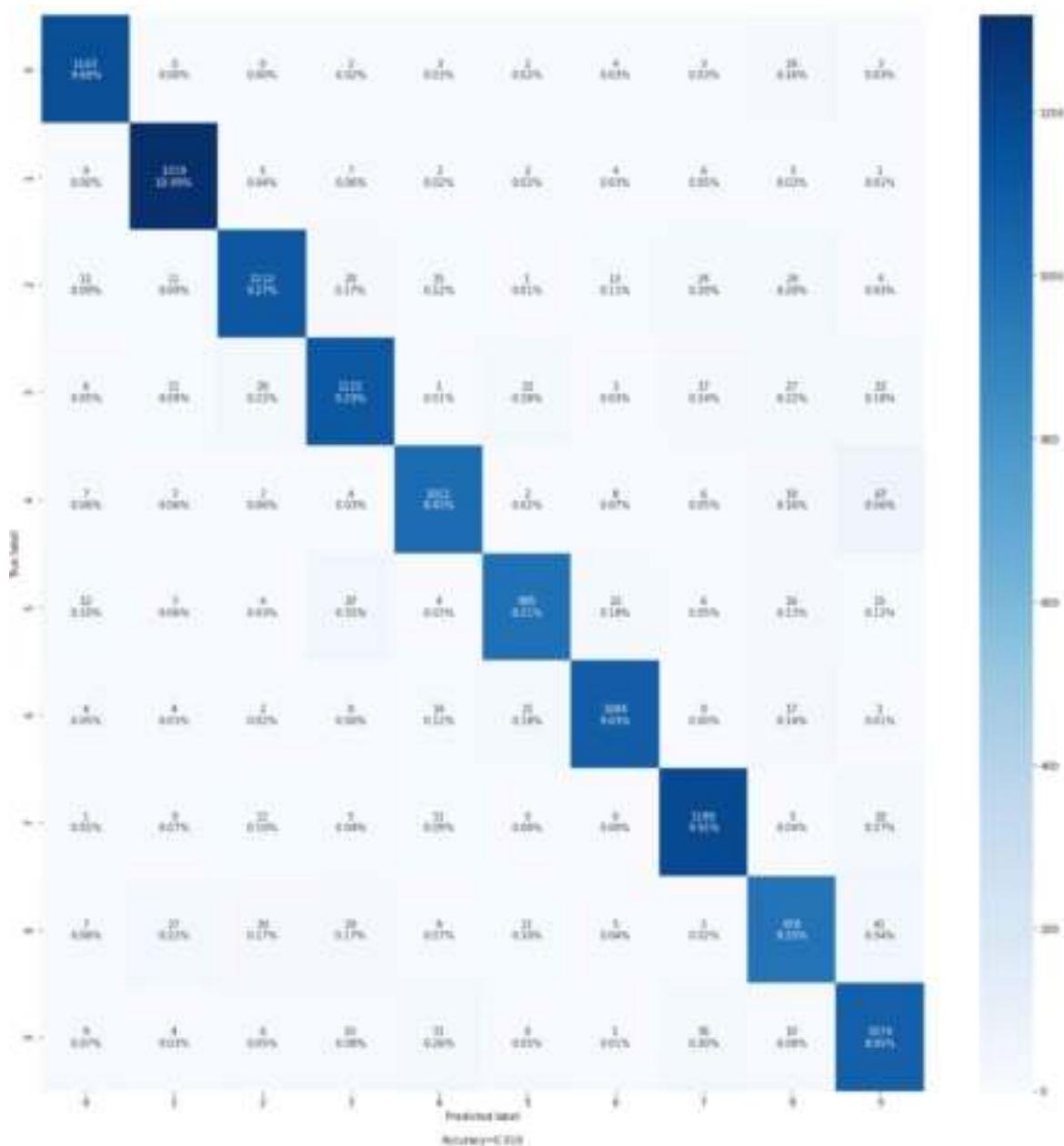
## Feature Importance

There are three different types of feature importance visualizations provided: Weight, Gain and Coverage. We provide detailed definitions for each of the three in the report. Feature importance visualizations help you learn what features in your training dataset contributed to the predictions. Feature importance visualizations are available for the following model types: binary classification, multiclassification, and regression.



## Confusion Matrix

This visualization is only applicable to binary and multiclass classification models. Accuracy alone might not be sufficient for evaluating the model performance. For some use cases, such as healthcare and fraud detection, it's also important to know the false positive rate and false negative rate. A confusion matrix gives you the additional dimensions for evaluating your model performance.



### Evaluation of the Confusion Matrix

This section provides you with more insights on the micro, macro, and weighted metrics on precision, recall, and F1-score for your model.

**Overall Accuracy**

Overall Accuracy: 0.919

**Micro Performance Metrics**

Performance metrics calculated globally by counting the total true positives, false negatives, and false positives.

Micro Precision: 0.919

Micro Recall: 0.919

Micro F1-score: 0.919

**Macro Performance Metrics**

Performance metrics calculated for each label, and find their unweighted mean.  
This does not take the class imbalance problem into account.

Macro Precision: 0.918

Macro Recall: 0.918

Macro F1-score: 0.918

**Weighted Performance Metrics**

Performance metrics calculated for each label and their average weighted by support  
(the number of true instances for each label).  
This extends the macro option to take the class imbalance into account.  
It might result in an F-score that is not between precision and recall.

Weighted Precision: 0.92

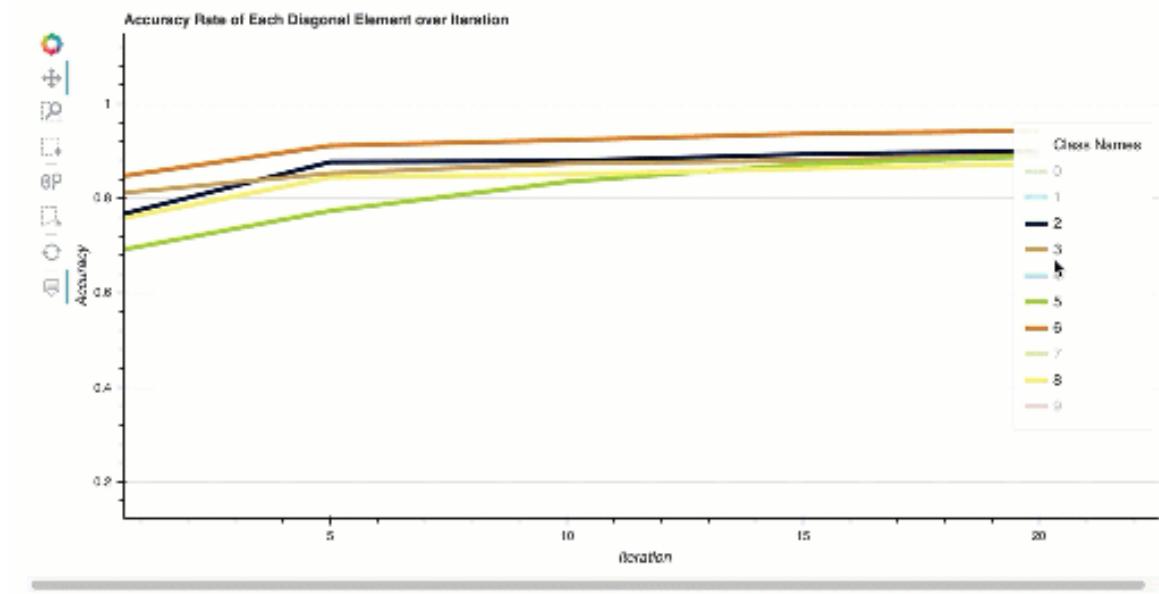
Weighted Recall: 0.919

Weighted F1-score: 0.919

**Classification Report**

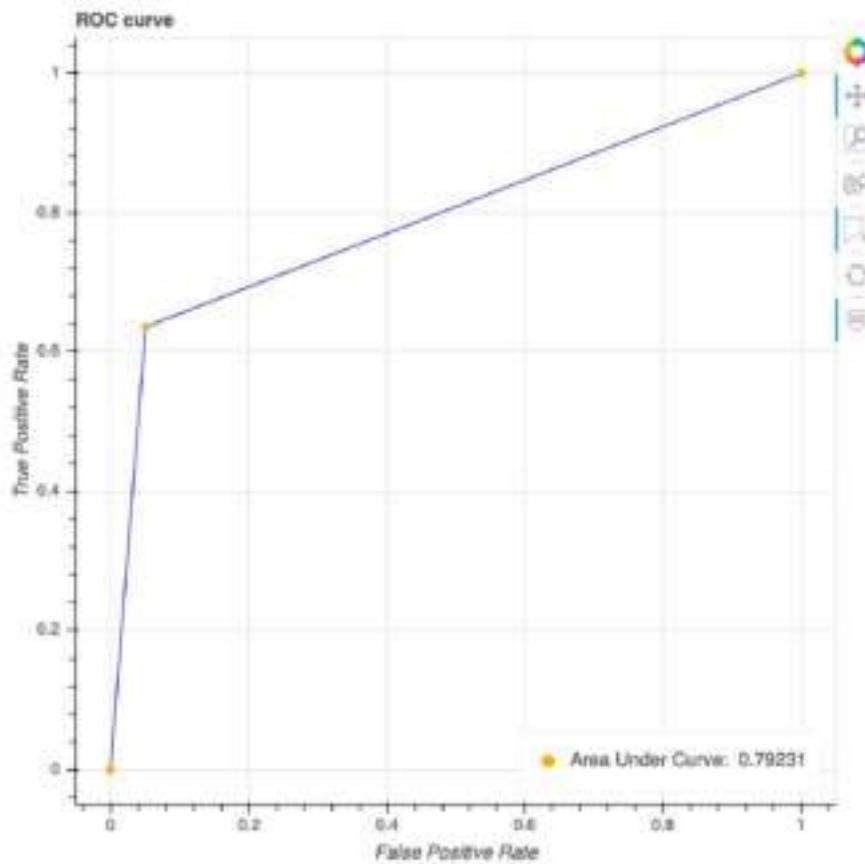
The summary of the precision, recall, and F1-score for each class.

	precision	recall	f1-score	support
0.0	0.95	0.97	0.96	1199
1.0	0.94	0.98	0.96	1349
2.0	0.93	0.98	0.92	1235
3.0	0.91	0.99	0.90	1250
4.0	0.92	0.99	0.90	1138
5.0	0.94	0.99	0.93	1108
6.0	0.95	0.94	0.95	1149
7.0	0.92	0.98	0.93	1264
8.0	0.87	0.87	0.87	1121
9.0	0.89	0.99	0.88	1187
accuracy			0.92	12000
macro avg	0.92	0.92	0.92	12000
weighted avg	0.92	0.92	0.92	12000



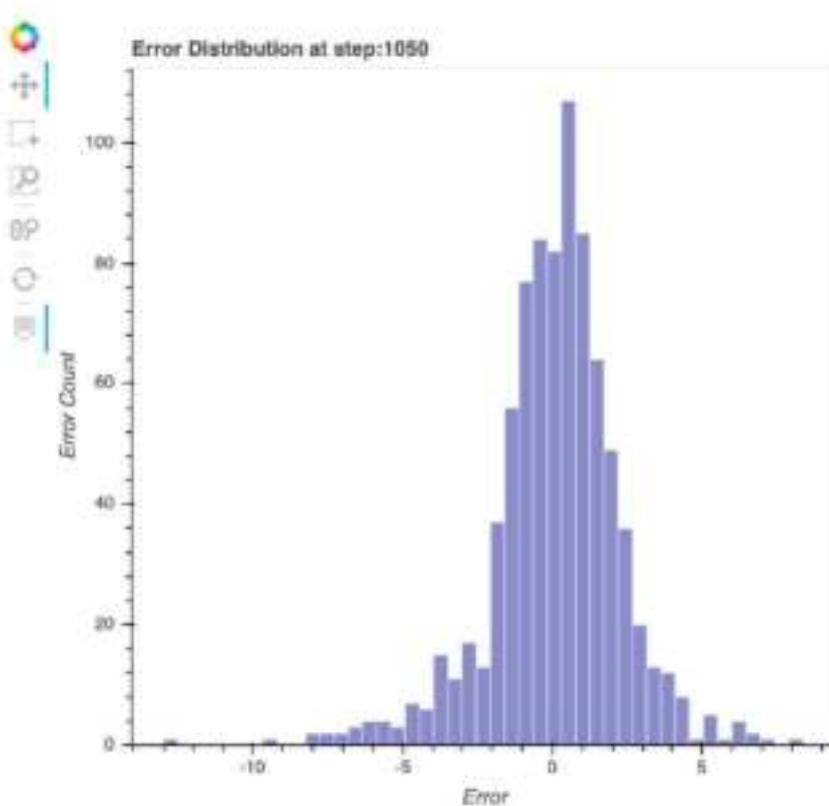
### Receiver Operating Characteristic Curve

This visualization is only applicable to binary classification models. The Receiver Operating Characteristic curve is commonly used to evaluate binary classification model performance. The y-axis of the curve is True Positive Rate (TPR) and x-axis is false positive rate (FPR). The plot also displays the value for the area under the curve (AUC). The higher the AUC value, the more predictive your classifier. You can also use the ROC curve to understand the trade-off between TPR and FPR and identify the optimum classification threshold for your use case. The classification threshold can be adjusted to tune the behavior of the model to reduce more of one or another type of error (FP/FN).



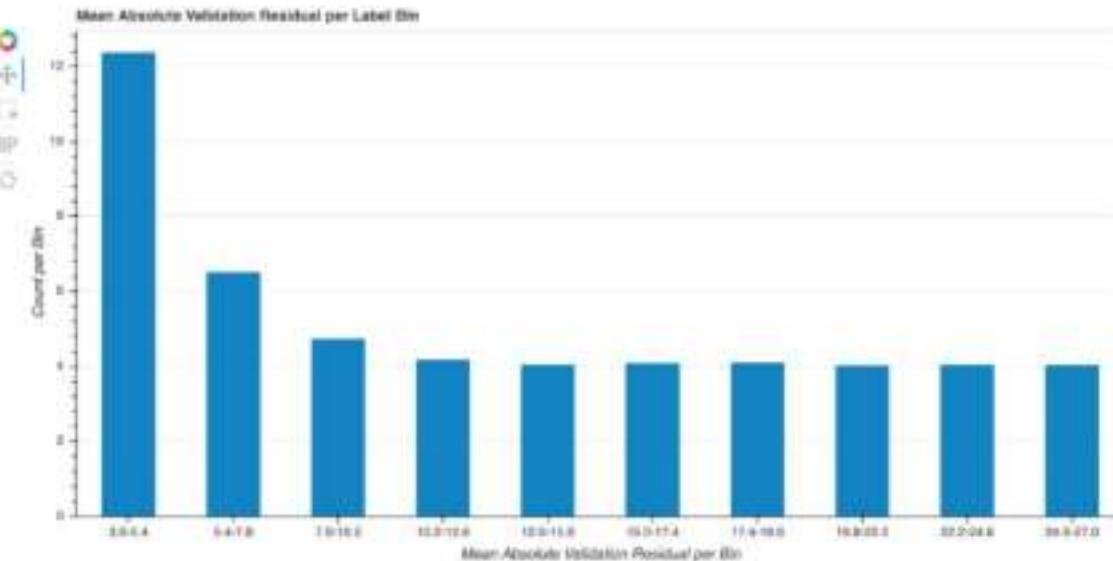
### Distribution of Residuals at the Last Saved Step

This visualization is a column chart that shows the residual distributions in the last step Debugger captures. In this visualization, you can check whether the residual distribution is close to normal distribution that's centered at zero. If the residuals are skewed, your features may not be sufficient for predicting the labels.



#### Absolute Validation Error per Label Bin Over Iteration

This visualization is only applicable to regression models. The actual target values are split into 10 intervals. This visualization shows how validation errors progress for each interval throughout the training steps in line plots. Absolute validation error is the absolute value of difference between prediction and actual during validation. You can identify the underperforming intervals from this visualization.



# Analyze Data Using the SMDebug Client Library

While your training job is running or after it has completed, you can access the training data collected by Debugger using the [Amazon SageMaker Python SDK](#) and the [SMDebug client library](#). The SMDebug library provides analysis and visualization tools that enable you to drill down into your training job data.

**To install the library and use the SMDebug analysis tools (in a JupyterLab notebook or an iPython kernel)**

```
! pip install -U smdebug
```

The following topics walk you through how to use the SMDebug tools to visualize and analyze the training data collected by Debugger.

## Analyze system and framework metrics

- [Access the Monitoring and Profiling Data \(p. 1724\)](#)
- [Plot the System Metrics and Framework Metrics Data \(p. 1725\)](#)
- [Access the Profiling Data Using the Pandas Data Parsing Tool \(p. 1726\)](#)
- [Access the Python Profiling Stats Data \(p. 1726\)](#)
- [Merge Timelines of Different Profiling Trace Files \(p. 1729\)](#)
- [Profiling Data Loader \(p. 1731\)](#)

## Access the Monitoring and Profiling Data

The SMDebug `TrainingJob` class reads data from the S3 bucket where the system and framework metrics are saved.

**To set up a `TrainingJob` object and retrieve profiling event files of a training job**

```
from smdebug.profiler.analysis.notebook_utils.training_job import TrainingJob
tj = TrainingJob(training_job_name, region)
```

### Tip

You need to specify the `training_job_name` and `region` parameters to log to a training job. There are two ways to specify the training job information:

- Use the SageMaker Python SDK while the estimator is still attached to the training job.

```
import sagemaker
training_job_name=estimator.latest_training_job.job_name
region=sagemaker.Session().boto_region_name
```

- Pass strings directly.

```
training_job_name="your-training-job-name-YYYY-MM-DD-HH-MM-SS-SSS"
region="us-west-2"
```

**To retrieve a description of the training job description and the S3 bucket URI where the metric data are saved**

```
tj.describe_training_job()
tj.get_config_and_profiler_s3_output_path()
```

### To check if the system and framework metrics are available from the S3 URI

```
tj.wait_for_sys_profiling_data_to_be_available()  
tj.wait_for_framework_profiling_data_to_be_available()
```

### To create system and framework reader objects after the metric data become available

```
system_metrics_reader = tj.get_systems_metrics_reader()  
framework_metrics_reader = tj.get_framework_metrics_reader()
```

### To refresh and retrieve the latest training event files

The reader objects have an extended method, `refresh_event_file_list()`, to retrieve the latest training event files.

```
system_metrics_reader.refresh_event_file_list()  
framework_metrics_reader.refresh_event_file_list()
```

## Plot the System Metrics and Framework Metrics Data

You can use the system and algorithm metrics objects for the following visualization classes to plot timeline graphs and histograms.

#### Note

To visualize the data with narrowed-down metrics in the following visualization object plot methods, specify `select_dimensions` and `select_events` parameters. For example, if you specify `select_dimensions=[ "GPU" ]`, the plot methods filter the metrics that include the "GPU" keyword. If you specify `select_events=[ "total" ]`, the plot methods filter the metrics that include the "total" event tags at the end of the metric names. If you enable these parameters and give the keyword strings, the visualization classes return the charts with filtered metrics.

- The `MetricsHistogram` class

```
from smdebug.profiler.analysis.notebook_utils.metrics_histogram import MetricsHistogram  
  
metrics_histogram = MetricsHistogram(system_metrics_reader)  
metrics_histogram.plot(  
    starttime=0,  
    endtime=system_metrics_reader.get_timestamp_of_latest_available_file(),  
    select_dimensions=[ "CPU", "GPU", "I/O" ], # optional  
    select_events=[ "total" ] # optional  
)
```

- The `StepTimelineChart` class

```
from smdebug.profiler.analysis.notebook_utils.step_timeline_chart import  
StepTimelineChart  
  
view_step_timeline_chart = StepTimelineChart(framework_metrics_reader)
```

- The `StepHistogram` class

```
from smdebug.profiler.analysis.notebook_utils.step_histogram import StepHistogram  
  
step_histogram = StepHistogram(framework_metrics_reader)  
step_histogram.plot(  
    starttime=step_histogram.last_timestamp - 5 * 1000 * 1000,
```

```
|     endtime=step_histogram.last_timestamp,  
|     show_workers=True  
)
```

- The `TimelineCharts` class

```
from smdebug.profiler.analysis.notebook_utils.timeline_charts import TimelineCharts  
  
view_timeline_charts = TimelineCharts(  
    system_metrics_reader,  
    framework_metrics_reader,  
    select_dimensions=["CPU", "GPU", "I/O"], # optional  
    select_events=["total"] # optional  
)  
  
view_timeline_charts.plot_detailed_profiler_data([700,710])
```

- The `Heatmap` class

```
from smdebug.profiler.analysis.notebook_utils.heatmap import Heatmap  
  
view_heatmap = Heatmap(  
    system_metrics_reader,  
    framework_metrics_reader,  
    select_dimensions=["CPU", "GPU", "I/O"], # optional  
    select_events=["total"], # optional  
    plot_height=450  
)
```

## Access the Profiling Data Using the Pandas Data Parsing Tool

The following `PandasFrame` class provides tools to convert the collected profiling data to Pandas data frame.

```
from smdebug.profiler.analysis.utils.profiler_data_to_pandas import PandasFrame
```

The `PandasFrame` class takes the `tj` object's S3 bucket output path, and its methods `get_all_system_metrics()` `get_all_framework_metrics()` return system metrics and framework metrics in the Pandas data format.

```
pf = PandasFrame(tj.profiler_s3_output_path)  
system_metrics_df = pf.get_all_system_metrics()  
framework_metrics_df = pf.get_all_framework_metrics()  
selected_framework_metrics=[  
    'Step:ModeKeys.TRAIN',  
    'Step:ModeKeys.GLOBAL'  
]  
)
```

## Access the Python Profiling Stats Data

The Python profiling provides framework metrics related to Python functions and operators in your training scripts and the SageMaker deep learning frameworks.

### Training Modes and Phases for Python Profiling

To profile specific intervals during training to partition statistics for each of these intervals, Debugger provides tools to set modes and phases.

For training modes, use the following `PythonProfileModes` class:

```
from smdebug.profiler.python_profile_utils import PythonProfileModes
```

This class provides the following options:

- `PythonProfileModes.TRAIN` – Use if you want to profile the target steps in the training phase. This mode option available only for TensorFlow.
- `PythonProfileModes.EVAL` – Use if you want to profile the target steps in the evaluation phase. This mode option available only for TensorFlow.
- `PythonProfileModes.PREDICT` – Use if you want to profile the target steps in the prediction phase. This mode option available only for TensorFlow.
- `PythonProfileModes.GLOBAL` – Use if you want to profile the target steps in the global phase, which includes the previous three phases. This mode option available only for PyTorch.
- `PythonProfileModes.PRE_STEP_ZERO` – Use if you want to profile the target steps in the initialization stage before the first training step of the first epoch starts. This phase includes the initial job submission, uploading the training scripts to EC2 instances, preparing the EC2 instances, and downloading input data. This mode option available for both TensorFlow and PyTorch.
- `PythonProfileModes.POST_HOOK_CLOSE` – Use if you want to profile the target steps in the finalization stage after the training job has done and the Debugger hook is closed. This phase includes profiling data while the training jobs are finalized and completed. This mode option available for both TensorFlow and PyTorch.

For training phases, use the following `StepPhase` class:

```
from smdebug.profiler.analysis.utils.python_profile_analysis_utils import StepPhase
```

This class provides the following options:

- `StepPhase.START` – Use to specify the start point of the initialization phase.
- `StepPhase.STEP_START` – Use to specify the start step of the training phase.
- `StepPhase.FORWARD_PASS_END` – Use to specify the steps where the forward pass ends. This option is available only for PyTorch.
- `StepPhase.STEP_END` – Use to specify the end steps in the training phase. This option is available only for TensorFlow.
- `StepPhase.END` – Use to specify the ending point of the finalization (post-hook-close) phase. If the callback hook is not closed, the finalization phase profiling does not occur.

## Python Profiling Analysis Tools

Debugger supports the Python profiling with two profiling tools:

- `cProfile` – The standard python profiler. `cProfile` collects framework metrics on CPU time for every function called when profiling was enabled.
- `Pyinstrument` – This is a low overhead Python profiler sampling profiling events every milliseconds.

To learn more about the Python profiling options and what's collected, see [Start a Training Job with the Default System Monitoring and Customized Framework Profiling with Different Profiling Options \(p. 1600\)](#).

The following methods of the `PythonProfileAnalysis`, `cProfileAnalysis`, `PyinstrumentAnalysis` classes are provided to fetch and analyze the Python profiling data. Each function loads the latest data from the default S3 URI.

```
from smdebug.profiler.analysis.python_profile_analysis import PythonProfileAnalysis,  
cProfileAnalysis, PyinstrumentAnalysis
```

To set Python profiling objects for analysis, use the `cProfileAnalysis` or `PyinstrumentAnalysis` classes as shown in the following example code. It shows how to set a `cProfileAnalysis` object, and if you want to use `PyinstrumentAnalysis`, replace the class name.

```
python_analysis = cProfileAnalysis(  
    local_profile_dir=tf_python_stats_dir,  
    s3_path=tj.profiler_s3_output_path  
)
```

The following methods are available for the `cProfileAnalysis` and `PyinstrumentAnalysis` classes to fetch the Python profiling stats data:

- `python_analysis.fetch_python_profile_stats_by_time(start_time_since_epoch_in_secs, end_time_since_epoch_in_secs)` – Takes in a start time and end time, and returns the function stats of step stats whose start or end times overlap with the provided interval.
- `python_analysis.fetch_python_profile_stats_by_step(start_step, end_step, mode, start_phase, end_phase)` – Takes in a start step and end step and returns the function stats of all step stats whose profiled step satisfies `start_step <= step < end_step`.
  - `start_step` and `end_step` (str) – Specify the start step and end step to fetch the Python profiling stats data.
  - `mode` (str) – Specify the mode of training job using the `PythonProfileModes` enumerator class. The default is `PythonProfileModes.TRAIN`. Available options are provided in the [Training Modes and Phases for Python Profiling \(p. 1726\)](#) section.
  - `start_phase` (str) – Specify the start phase in the target step(s) using the `StepPhase` enumerator class. This parameter enables profiling between different phases of training. The default is `StepPhase.STEP_START`. Available options are provided in the [Training Modes and Phases for Python Profiling \(p. 1727\)](#) section.
  - `end_phase` (str) – Specify the end phase in the target step(s) using the `StepPhase` enumerator class. This parameter sets up the end phase of training. Available options are as same as the ones for the `start_phase` parameter. The default is `StepPhase.STEP_END`. Available options are provided in the [Training Modes and Phases for Python Profiling \(p. 1727\)](#) section.
- `python_analysis.fetch_profile_stats_between_modes(start_mode, end_mode)` – Fetches stats from the Python profiling between the start and end modes.
- `python_analysis.fetch_pre_step_zero_profile_stats()` – Fetches the stats from the Python profiling until step 0.
- `python_analysis.fetch_post_hook_close_profile_stats()` – Fetches stats from the Python profiling after the hook is closed.
- `python_analysis.list_profile_stats()` – Returns a DataFrame of the Python profiling stats. Each row holds the metadata for each instance of profiling and the corresponding stats file (one per step).
- `python_analysis.list_available_node_ids()` – Returns a list the available node IDs for the Python profiling stats.

The `cProfileAnalysis` class specific methods:

- `fetch_profile_stats_by_training_phase()` – Fetches and aggregates the Python profiling stats for every possible combination of start and end modes. For example, if a training and validation phases are done while detailed profiling is enabled, the combinations are `(PRE_STEP_ZERO, TRAIN)`, `(TRAIN, TRAIN)`, `(TRAIN, EVAL)`, `(EVAL, EVAL)`, and `(EVAL, POST_HOOK_CLOSE)`. All stats files within each of these combinations are aggregated.

- `fetch_profile_stats_by_job_phase()` – Fetches and aggregates the Python profiling stats by job phase. The job phases are `initialization` (profiling until step 0), `training_loop` (training and validation), and `finalization` (profiling after the hook is closed).

## Merge Timelines of Different Profiling Trace Files

The SMDebug client library provide profiling analysis and visualization tools for merging timelines of system metrics, framework metrics, and Python profiling data collected by Debugger.

### Tip

Before proceeding, you need to set a `TrainingJob` object that will be utilized throughout the examples in this page. For more information about setting up a `TrainingJob` object, see [Access the Monitoring and Profiling Data \(p. 1724\)](#).

The `MergedTimeline` class provides tools to integrate and correlate different profiling information in a single timeline. After Debugger captures profiling data and annotations from different phases of a training job, JSON files of trace events are saved in a default `tracefolder` directory.

- For annotations in the Python layers, the trace files are saved in `*pythontimeline.json`.
- For annotations in the TensorFlow C++ layers, the trace files are saved in `*model_timeline.json`.
- Tensorflow profiler saves events in a `*trace.json.gz` file.

### Tip

If you want to list all of the JSON trace files, use the following Amazon CLI command:

```
! aws s3 ls {tj.profiler_s3_output_path} --recursive | grep '\.json$'
```

As shown in the following animated screenshot, putting and aligning the trace events captured from the different profiling sources in a single plot can provide an overview of the entire events occurring in different phases of the training job.



### Tip

To interact with the merged timeline on the traicing app using a keyboard, use the **w** key for zooming in, the **A** key for shifting to the left, the **S** key for zooming out, and the **D** key for shifting to the right.

The multiple event trace JSON files can be merged into one trace event JSON file using the following `MergedTimeline` API operation and class method from the `smdebug.profiler.analysis.utils.merge_timelines` module.

```
from smdebug.profiler.analysis.utils.merge_timelines import MergedTimeline

combined_timeline = MergedTimeline(path, file_suffix_filter, output_directory)
combined_timeline.merge_timeline(start, end, unit)
```

The `MergedTimeline` API operation passes the following parameters:

- **path (str)** – Specify a root folder (`/profiler-output`) that contains system and framework profiling trace files. You can locate the `profiler-output` using the SageMaker estimator classmethod or the `TrainingJob` object. For example, `estimator.latest_job_profiler_artifacts_path()` or `tj.profiler_s3_output_path`.
- **file\_suffix\_filter (list)** – Specify a list of file suffix filters to merge timelines. Available suffix filters are `["model_timeline.json", "pythontimeline.json", "trace.json.gz"]`. If this parameter is not manually specified, all of the trace files are merged by default.
- **output\_directory (str)** – Specify a path to save the merged timeline JSON file. The default is to the directory specified for the `path` parameter.

The `merge_timeline()` classmethod passes the following parameters to execute the merging process:

- **start (int)** – Specify start time (in microseconds and in Unix time format) or start step to merge timelines.
- **end (int)** – Specify end time (in microseconds and in Unix time format) or end step to merge timelines.
- **unit (str)** – Choose between `"time"` and `"step"`. The default is `"time"`.

Using the following example codes, execute the `merge_timeline()` method and download the merged JSON file.

- Merge timeline with the `"time"` unit option. The following example code merges all available trace files between the Unix start time (the absolute zero Unix time) and the current Unix time, which means that you can merge the timelines for the entire training duration.

```
import time
from smdebug.profiler.analysis.utils.merge_timelines import MergedTimeline
from smdebug.profiler_constants import CONVERT_TO_MICROSECS

combined_timeline = MergedTimeline(tj.profiler_s3_output_path, output_directory="./")
combined_timeline.merge_timeline(0, int(time.time() * CONVERT_TO_MICROSECS))
```

- Merge timeline with the `"step"` unit option. The following example code merges all available timelines between step 3 and step 9.

```
from smdebug.profiler.analysis.utils.merge_timelines import MergedTimeline

combined_timeline = MergedTimeline(tj.profiler_s3_output_path, output_directory="./")
combined_timeline.merge_timeline(3, 9, unit="step")
```

Open the Chrome tracing app at <chrome://tracing> on a Chrome browser, and open the JSON file. You can explore the output to plot the merged timeline.

## Profiling Data Loader

In PyTorch, data loader iterators, such as `SingleProcessingDataLoaderIter` and `MultiProcessingDataLoaderIter`, are initiated at the beginning of every iteration over a dataset. During the initialization phase, PyTorch turns on worker processes depending on the configured number of workers, establishes data queue to fetch data and `pin_memory` threads.

To use the PyTorch data loader profiling analysis tool, import the following `PT_dataloader_analysis` class:

```
from smdebug.profiler.analysis.utils.pytorch_dataloader_analysis import  
PT_dataloader_analysis
```

Pass the profiling data retrieved as a Pandas frame data object in the [Access the Profiling Data Using the Pandas Data Parsing Tool \(p. 1726\)](#) section:

```
pt_analysis = PT_dataloader_analysis(pf)
```

The following functions are available for the `pt_analysis` object:

The `SMDebug S3SystemMetricsReader` class reads the system metrics from the S3 bucket specified to the `s3_trial_path` parameter.

- `pt_analysis.analyze_dataloaderIter_initialization()`

The analysis outputs the median and maximum duration for these initializations. If there are outliers, (i.e duration is greater than  $2 * \text{median}$ ), the function prints the start and end times for those durations. These can be used to inspect system metrics during those time intervals.

The following list shows what analysis is available from this class method:

- Which type of data loader iterators were initialized.
  - The number of workers per iterator.
  - Inspect whether the iterator was initialized with or without `pin_memory`.
  - Number of times the iterators were initialized during training.
- `pt_analysis.analyze_dataloaderWorkers()`

The following list shows what analysis is available from this class method:

- The number of worker processes that were spun off during the entire training.
- Median and maximum duration for the worker processes.
- Start and end time for the worker processes that are outliers.

- `pt_analysis.analyze_dataloader_getnext()`

The following list shows what analysis is available from this class method:

- Number of GetNext calls made during the training.
  - Median and maximum duration in microseconds for GetNext calls.
  - Start time, End time, duration and worker id for the outlier GetNext call duration.
- `pt_analysis.analyze_batchtime(start_timestamp, end_timestamp, select_events=['.*'], select_dimensions=['.*'])`

Debugger collects the start and end times of all the GetNext calls. You can find the amount of time spent by the training script on one batch of data. Within the specified time window, you can identify the calls that are not directly contributing to the training. These calls can be from the following

operations: computing the accuracy, adding the losses for debugging or logging purposes, and printing the debugging information. Operations like these can be compute intensive or time consuming. We can identify such operations by correlating the Python profiler, system metrics, and framework metrics.

The following list shows what analysis is available from this class method:

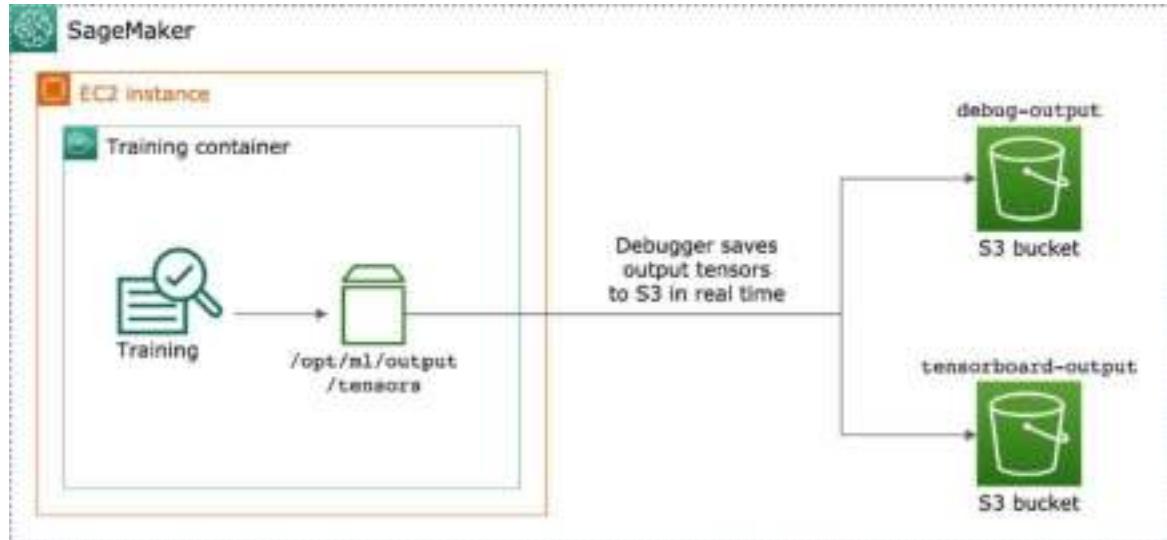
- Profile time spent on each data batch, `BatchTime_in_seconds`, by finding the difference between start times of current and subsequent `GetNext` calls.
- Find the outliers in `BatchTime_in_seconds` and start and end time for those outliers.
- Obtain the system and framework metrics during those `BatchTime_in_seconds` timestamps. This indicates where the time was spent.

• `pt_analysis.plot_the_window()`

Plots a timeline charts between a start timestamp and the end timestamp.

## Visualize Amazon SageMaker Debugger Output Tensors in TensorBoard

Use SageMaker Debugger to create output tensor files that are compatible with TensorBoard. Load the files to visualize in TensorBoard and analyze your SageMaker training jobs. Debugger automatically generates output tensor files that are compatible with TensorBoard. For any hook configuration you customize for saving output tensors, Debugger has the flexibility to create scalar summaries, distributions, and histograms that you can import to TensorBoard.



You can enable this by passing `DebuggerHookConfig` and `TensorBoardOutputConfig` objects to an estimator.

The following procedure explains how to save scalars, weights, and biases as full tensors, histograms, and distributions that can be visualized with TensorBoard. Debugger saves them to the training container's local path (the default path is `/opt/ml/output/tensors`) and syncs to the Amazon S3 locations passed through the Debugger output configuration objects.

### To save TensorBoard compatible output tensor files using Debugger

1. Set up a `tensorboard_output_config` configuration object to save TensorBoard output using the Debugger `TensorBoardOutputConfig` class. For the `s3_output_path` parameter, specify the

default S3 bucket of the current SageMaker session or a preferred S3 bucket. This example does not add the `container_local_output_path` parameter; instead, it is set to the default local path `/opt/ml/output/tensors`.

```
import sagemaker
from sagemaker.debugger import TensorBoardOutputConfig

bucket = sagemaker.Session().default_bucket()
tensorboard_output_config = TensorBoardOutputConfig(
    s3_output_path='s3://{}'.format(bucket)
)
```

For additional information, see the Debugger [TensorBoardOutputConfig](#) API in the [Amazon SageMaker Python SDK](#).

2. Configure the Debugger hook and customize the hook parameter values. For example, the following code configures a Debugger hook to save all scalar outputs every 100 steps in training phases and 10 steps in validation phases, the weights parameters every 500 steps (the default `save_interval` value for saving tensor collections is 500), and the biases parameters every 10 global steps until the global step reaches 500.

```
from sagemaker.debugger import CollectionConfig, DebuggerHookConfig

hook_config = DebuggerHookConfig(
    hook_parameters={
        "train.save_interval": "100",
        "eval.save_interval": "10"
    },
    collection_configs=[
        CollectionConfig("weights"),
        CollectionConfig(
            name="biases",
            parameters={
                "save_interval": "10",
                "end_step": "500",
                "save_histogram": "True"
            }
        ),
    ]
)
```

For more information about the Debugger configuration APIs, see the Debugger [CollectionConfig](#) and [DebuggerHookConfig](#) APIs in the [Amazon SageMaker Python SDK](#).

3. Construct a SageMaker estimator with the Debugger parameters passing the configuration objects. The following example template shows how to create a generic SageMaker estimator. You can replace `estimator` and `Estimator` with other SageMaker frameworks' estimator parent classes and estimator classes. Available SageMaker framework estimators for this functionality are [TensorFlow](#), [PyTorch](#), and [MXNet](#).

```
from sagemaker.estimator import Estimator

estimator = Estimator(
    ...
    # Debugger parameters
    debugger_hook_config=hook_config,
    tensorboard_output_config=tensorboard_output_config
)
estimator.fit()
```

The estimator.fit() method starts a training job, and Debugger writes the output tensor files in real time to the Debugger S3 output path and to the TensorBoard S3 output path. To retrieve the output paths, use the following estimator methods:

- For the Debugger S3 output path, use estimator.latest\_job\_debugger\_artifacts\_path().
  - For the TensorBoard S3 output path, use estimator.latest\_job\_tensorboard\_artifacts\_path().
4. After the training has completed, check the names of saved output tensors:

```
from smdebug.trials import create_trial
trial = create_trial(estimator.latest_job_debugger_artifacts_path())
trial.tensor_names()
```

5. Check the TensorBoard output data in Amazon S3:

```
tensorboard_output_path=estimator.latest_job_tensorboard_artifacts_path()
print(tensorboard_output_path)
!aws s3 ls {tensorboard_output_path}/
```

6. Download the TensorBoard output data to your notebook instance. For example, the following Amazon CLI command downloads the TensorBoard files to /logs/fit under the current working directory of your notebook instance.

```
!aws s3 cp --recursive {tensorboard_output_path} ./logs/fit
```

7. Compress the file directory to a TAR file to download to your local machine.

```
!tar -cf logs.tar logs
```

8. Download and extract the Tensorboard TAR file to a directory on your device, launch a Jupyter notebook server, open a new notebook, and run the TensorBoard app.

```
!tar -xf logs.tar
%load_ext tensorboard
%tensorboard --logdir logs/fit
```

## Best Practices for Amazon SageMaker Debugger

Use the following guidelines when you run training jobs with Debugger.

### Topics

- [Choose a Machine Learning Framework \(p. 1735\)](#)
- [Use Studio Debugger Insights Dashboard \(p. 1735\)](#)
- [Download Debugger Reports and Gain More Insights \(p. 1735\)](#)
- [Capture Data from Your Training Job and Save Data to Amazon S3 \(p. 1735\)](#)
- [Analyze the Data with a Fleet of Debugger Built-in Rules \(p. 1735\)](#)
- [Take Actions Based on the Built-in Rule Status \(p. 1735\)](#)
- [Dive Deep into the Data Using the SMDebug Client Library \(p. 1736\)](#)
- [Monitoring System Utilization and Detect Bottlenecks \(p. 1736\)](#)
- [Profiling Framework Operations \(p. 1736\)](#)

- [Debugging Model Parameters \(p. 1736\)](#)

## Choose a Machine Learning Framework

You can choose a machine learning framework and use SageMaker pre-built training containers or your own containers. Use Debugger to detect training and performance issues, and analyze training progress of your training job in SageMaker. SageMaker provides you options to use pre-built containers that are prepared for a number of machine learning framework environments to train your model on Amazon EC2. Any training job can be adapted to run in Amazon Deep Learning Containers, SageMaker training containers, and custom containers. To learn more, see [Configure Debugger Using Amazon SageMaker Python SDK \(p. 1593\)](#) and [Use Debugger with Custom Training Containers \(p. 1672\)](#).

## Use Studio Debugger Insights Dashboard

With Studio Debugger insights dashboard, you are in control of your training jobs. Use the Studio Debugger dashboards to keep your model performance on Amazon EC2 instances in control and optimized. For any SageMaker training jobs running on Amazon EC2 instance, Debugger monitors resource utilization and basic model output data (loss and accuracy values). Through the Studio Debugger dashboards, gain insights into your training jobs and improve your model training performance. To learn more, see [SageMaker Debugger on Studio \(p. 1685\)](#).

## Download Debugger Reports and Gain More Insights

You can view aggregated results and gain insights in Debugger reports. Debugger aggregates training and profiling results collected from the built-in rule analysis into a report per training job. You can find more detailed information about your training results through the Debugger reports. To learn more, see [SageMaker Debugger Interactive Reports \(p. 1700\)](#).

## Capture Data from Your Training Job and Save Data to Amazon S3

You can use a Debugger hook to save output tensors. After you choose a container and a framework that fit your training script, use a Debugger hook to configure which tensors to save and to which directory to save them, such as a Amazon S3 bucket. A Debugger hook helps you to build the configuration and to keep it in your account to use in subsequent analyses, where it is secured for use with the most privacy-sensitive applications. To learn more, see [Configure Debugger Hook to Save Tensors \(p. 1602\)](#).

## Analyze the Data with a Fleet of Debugger Built-in Rules

You can use Debugger built-in rules to inspect tensors in parallel with a training job. To analyze the training performance data, Debugger provides built-in rules that watch for abnormal training process behaviors. For example, a Debugger rule detects issues when the training process suffers from system bottleneck issues or training issues, such as vanishing gradients, exploding tensors, overfitting, or overtraining. If necessary, you can also build customized rules by creating a rule definition with your own criteria to define a training issue. To learn more about the Debugger rules, see [Configure Debugger Built-in Rules \(p. 1608\)](#) for detailed instructions of using the [Amazon SageMaker Python SDK](#). For a full list of the Debugger built-in rules, see [List of Debugger Built-in Rules \(p. 1626\)](#). If you want to create a custom rule, see [Create Debugger Custom Rules for Training Job Analysis \(p. 1670\)](#).

## Take Actions Based on the Built-in Rule Status

You can use Debugger with Amazon CloudWatch Events and Amazon Lambda. You can automate actions based on the rule status, such as stopping training jobs early and setting up notifications through email or text. When the Debugger rules detect problems and triggers an "IssuesFound" evaluation status,

CloudWatch Events detects the rule status changes and invokes the Lambda function to take actions. To configure automated actions to your training issues, see [Create Actions on Rules Using Amazon CloudWatch and Amazon Lambda \(p. 1680\)](#).

## Dive Deep into the Data Using the SMDebug Client Library

You can use the SMDebug tools to access and analyze training data collected by Debugger. The `TrainingJob` and `create_trial` classes load the metrics and tensors saved by Debugger. These classes provide extended class methods to analyze the data in real time or after the training has finished. The SMDebug library also provides visualization tools: merge timelines of framework metrics to aggregate different profiling, line charts and heatmap to track the system utilization, and histograms to find step duration outliers. To learn more about the SMDebug library tools, see [Analyze Data Using the SMDebug Client Library \(p. 1724\)](#).

## Monitoring System Utilization and Detect Bottlenecks

With Amazon SageMaker Debugger monitoring, you can measure hardware system resource utilization of Amazon EC2 instances. Monitoring is available for any SageMaker training job constructed with the SageMaker framework estimators (TensorFlow, PyTorch, and MXNet) and the generic SageMaker estimator (SageMaker built-in algorithms and your own custom containers). Debugger built-in rules for monitoring detect system bottleneck issues and notify you when they detect the bottleneck issues.

To learn how to enable Debugger system monitoring, see [Configure Debugger Using Amazon SageMaker Python SDK \(p. 1593\)](#) and then [Configure Debugger Monitoring Hardware System Resource Utilization \(p. 1598\)](#).

For a full list of available built-in rules for monitoring, see [Debugger Built-in Rules for Monitoring Hardware System Resource Utilization \(System Metrics\) \(p. 1626\)](#).

## Profiling Framework Operations

With Amazon SageMaker Debugger profiling you can profile deep learning frameworks operations. You can profile your model training with the SageMaker TensorFlow training containers, the SageMaker PyTorch framework containers, and your own training containers. Using the profiling feature of Debugger, you can drill down into the Python operators and functions that are executed to perform the training job. Debugger supports detailed profiling, Python profiling, data loader profiling, and Horovod distributed training profiling. You can merge the profiled timelines to correlate with the system bottlenecks. Debugger built-in rules for profiling watch framework operation related issues, including excessive training initialization time due to data downloading before training starts and step duration outliers in training loops.

To learn how to configure Debugger for framework profiling, see [Configure Debugger Using Amazon SageMaker Python SDK \(p. 1593\)](#) and then [Configure Debugger Framework Profiling \(p. 1598\)](#).

For a complete list of available built-in rules for profiling, see [Debugger Built-in Rules for Profiling Framework Metrics \(p. 1626\)](#).

## Debugging Model Parameters

Debugging is available for deep learning frameworks using Amazon Deep Learning Containers and the SageMaker training containers. For fully supported framework versions (see the versions at [Supported Frameworks and Algorithms \(p. 1581\)](#)), Debugger automatically registers hooks to collect output tensors, and you can directly run your training script. For the versions with one asterisk sign, you need to manually register the hooks to collect tensors. Debugger provides preconfigured tensor collections with generalized names that you can utilize across the different frameworks. If you want to customize output tensor configuration, you can also use the `CollectionConfig` and `DebuggerHookConfig` API operations and the [Amazon SageMaker Python SDK](#) to configure your own tensor collections. Debugger built-in rules

for debugging analyze the output tensors and identifies model optimization problems that blocks your model from minimizing the loss function. For example, the rules identify overfitting, overtraining, loss not decreasing, exploding tensors, and vanishing gradients.

To learn how to configure Debugger for debugging output tensors, see [Configure Debugger Using Amazon SageMaker Python SDK \(p. 1593\)](#) and then [Configure Debugger Hook to Save Tensors \(p. 1602\)](#).

For a full list of available built-in rules for debugging, see [Debugger Built-in Rules for Debugging Model Training Data \(Output Tensors\) \(p. 1627\)](#).

## Amazon SageMaker Debugger Advanced Topics and Reference Documentation

The following sections contain advanced topics, reference documentation for the API operations, exceptions, and known limitations for Debugger.

### Topics

- [Amazon SageMaker Debugger API Operations \(p. 1737\)](#)
- [Use Debugger Docker Images for Built-in or Custom Rules \(p. 1738\)](#)
- [Amazon SageMaker Debugger Exceptions \(p. 1740\)](#)
- [Considerations for Amazon SageMaker Debugger \(p. 1741\)](#)
- [Amazon SageMaker Debugger Usage Statistics \(p. 1743\)](#)

## Amazon SageMaker Debugger API Operations

Amazon SageMaker Debugger has API operations in several locations that are used to implement its monitoring and analysis of model training.

Amazon SageMaker Debugger also provides the open source SMDebug Python library at [awslabs/sagemaker-debugger](#) that is used to configure built-in rules, define custom rules, and register hooks to collect output tensor data from training jobs.

The [Amazon SageMaker Python SDK](#) is a high-level SDK focused on machine learning experimentation. The SDK can be used to deploy built-in or custom rules defined with the SMDebug Python library to monitor and analyze these tensors using SageMaker estimators.

Debugger has added operations and types to the Amazon SageMaker API that enable the platform to use Debugger when training a model and to manage the configuration of inputs and outputs.

- `CreateTrainingJob` and `UpdateTrainingJob` use the following Debugger APIs to configure tensor collections, rules, rule images, and profiling options:
  - `CollectionConfiguration`
  - `DebugHookConfig`
  - `DebugRuleConfiguration`
  - `TensorBoardOutputConfig`
  - `ProfilerConfig`
  - `ProfilerRuleConfiguration`
- `DescribeTrainingJob` provides a full description of a training job, including the following Debugger configurations and rule evaluation statuses:
  - `DebugHookConfig`
  - `DebugRuleConfiguration`
  - `DebugRuleEvaluationStatus`

- [ProfilerConfig](#)
- [ProfilerRuleConfiguration](#)
- [ProfilerRuleEvaluationStatus](#)

The rule configuration API operations use the SageMaker Processing functionality when analyzing a model training. For more information about SageMaker Processing, see [Process Data \(p. 664\)](#).

## Use Debugger Docker Images for Built-in or Custom Rules

Amazon SageMaker provides two sets of Docker images for rules: one set for evaluating rules provided by SageMaker (built-in rules) and one set for evaluating custom rules provided in Python source files.

If you use the [Amazon SageMaker Python SDK](#), you can simply use SageMaker high-level Debugger API operations with SageMaker Estimator API operations, without having to manually retrieve the Debugger Docker images and configure the `ConfigureTrainingJobAPI`.

If you are not using the SageMaker Python SDK, you have to retrieve a relevant pre-built container base image for the Debugger rules. Amazon SageMaker Debugger provides pre-built Docker images for built-in and custom rules, and the images are stored in Amazon Elastic Container Registry (Amazon ECR). To pull an image from an Amazon ECR repository (or to push an image to one), use the full name registry URL of the image using the `CreateTrainingJob` API. SageMaker uses the following URL patterns for the Debugger rule container image registry address.

```
<account_id>.dkr.ecr.<Region>.amazonaws.com/<ECR repository name>:<tag>
```

For the account ID in each Amazon Region, Amazon ECR repository name, and tag value, see the following topics.

### Topics

- [Amazon SageMaker Debugger Registry URLs for Built-in Rule Evaluators \(p. 1738\)](#)
- [Amazon SageMaker Debugger Registry URLs for Custom Rule Evaluators \(p. 1739\)](#)

## Amazon SageMaker Debugger Registry URLs for Built-in Rule Evaluators

Use the following values for the components of the registry URLs for the images that provide built-in rules for Amazon SageMaker Debugger. For account IDs, see the following table.

**ECR Repository Name:** sagemaker-debugger-rules

**Tag:** latest

**Example of a full registry URL:**

```
904829902805.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-debugger-rules:latest
```

### Account IDs for Built-in Rules Container Images by Amazon Region

Region	account_id
af-south-1	314341159256
ap-east-1	199566480951
ap-northeast-1	430734990657
ap-northeast-2	578805364391

Region	account_id
ap-south-1	904829902805
ap-southeast-1	972752614525
ap-southeast-2	184798709955
ca-central-1	519511493484
cn-north-1	618459771430
cn-northwest-1	658757709296
eu-central-1	482524230118
eu-north-1	314864569078
eu-south-1	563282790590
eu-west-1	929884845733
eu-west-2	250201462417
eu-west-3	447278800020
me-south-1	986000313247
sa-east-1	818342061345
us-east-1	503895931360
us-east-2	915447279597
us-west-1	685455198987
us-west-2	895741380848
us-gov-west-1	515509971035

## Amazon SageMaker Debugger Registry URLs for Custom Rule Evaluators

Use the following values for the components of the registry URL for the images that provide custom rule evaluators for Amazon SageMaker Debugger. For account IDs, see the following table.

**ECR Repository Name:** sagemaker-debugger-rule-evaluator

**Tag:** latest

**Example of a full registry URL:**

552407032007.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-debugger-rule-evaluator:latest

## Account IDs for Custom Rules Container Images by Amazon Region

Region	account_id
af-south-1	515950693465
ap-east-1	645844755771

Region	account_id
ap-northeast-1	670969264625
ap-northeast-2	326368420253
ap-south-1	552407032007
ap-southeast-1	631532610101
ap-southeast-2	445670767460
ca-central-1	105842248657
cn-north-1	617202126805
cn-northwest-1	658559488188
eu-central-1	691764027602
eu-north-1	091235270104
eu-south-1	335033873580
eu-west-1	606966180310
eu-west-2	074613877050
eu-west-3	224335253976
me-south-1	050406412588
sa-east-1	466516958431
us-east-1	864354269164
us-east-2	840043622174
us-west-1	952348334681
us-west-2	759209512951
us-gov-west-1	515361955729

## Amazon SageMaker Debugger Exceptions

Amazon SageMaker Debugger is designed to be aware that tensors required to execute a rule might not be available at every step. As a result, it raises a few exceptions, which enable you to control what happens when a tensor is missing. These exceptions are available in the [smdebug.exceptions module](#). You can import them as follows:

```
from smdebug.exceptions import *
```

The following exceptions are available:

- **TensorUnavailableForStep** – The tensor requested is not available for the step. This might mean that this step might not be saved at all by the hook, or that this step might have saved some tensors but the requested tensor is not part of them. Note that when you see this exception, it means that this tensor can never become available for this step in the future. If the tensor has reductions saved for the step, it notifies you they can be queried.

- `TensorUnavailable` – This tensor is not being saved or has not been saved by the `smdebug` API. This means that this tensor is never seen for any step in `smdebug`.
- `StepUnavailable` – The step was not saved and Debugger has no data from the step.
- `StepNotYetAvailable` – The step has not yet been seen by `smdebug`. It might be available in the future if the training is still going on. Debugger automatically loads new data as it becomes available.
- `NoMoreData` – Raised when the training ends. Once you see this, you know that there are no more steps and no more tensors to be saved.
- `IndexReaderException` – The index reader is not valid.
- `InvalidWorker` – A worker was invoked that was not valid.
- `RuleEvaluationConditionMet` – Evaluation of the rule at the step resulted in the condition being met.
- `InsufficientInformationForRuleInvocation` – Insufficient information was provided to invoke the rule.

## Considerations for Amazon SageMaker Debugger

Consider the following when using Amazon SageMaker Debugger.

### Considerations for Distributed Training

The following list shows the scope of validity and considerations for using Debugger on training jobs with deep learning frameworks and various distributed training options.

- **Horovod**

#### Scope of validity of using Debugger for training jobs with Horovod

Deep Learning Framework	Apache MXNet	TensorFlow 1.x	TensorFlow 2.x	TensorFlow 2.x with Keras	PyTorch
Monitoring system bottlenecks	Yes	Yes	Yes	Yes	Yes
Profiling framework operations	No	No	No	Yes	Yes
Debugging model output tensors	Yes	Yes	Yes	Yes	Yes

- **SageMaker distributed data parallel**

#### Scope of validity of using Debugger for training jobs with SageMaker distributed data parallel

Deep Learning Framework	TensorFlow 2.x	TensorFlow 2.x with Keras	PyTorch
Monitoring system bottlenecks	Yes	Yes	Yes
Profiling framework operations	No*	No**	Yes

Deep Learning Framework	TensorFlow 2.x	TensorFlow 2.x with Keras	PyTorch
Debugging model output tensors	Yes	Yes	Yes

\* Debugger does not support framework profiling for TensorFlow 2.x.

\*\* SageMaker distributed data parallel does not support TensorFlow 2.x with Keras implementation.

- **SageMaker distributed model parallel** – Debugger does not support SageMaker distributed model parallel training.
- **Distributed training with SageMaker checkpoints** – Debugger is not available for training jobs when both the distributed training option and SageMaker checkpoints are enabled. You might see an error that looks like the following:

```
SMDebug Does Not Currently Support Distributed Training Jobs With Checkpointing Enabled
```

To use Debugger for training jobs with distributed training options, you need to disable SageMaker checkpointing and add manual checkpointing functions to your training script. For more information about using Debugger with distributed training options and checkpoints, see [Considerations for Using SageMaker Distributed Data Parallel with SageMaker Debugger and Checkpoints \(p. 1794\)](#) and [Saving Checkpoints \(p. 1818\)](#).

- **Parameter Server** – Debugger does not support parameter server-based distributed training.
- Profiling distributed training framework operations, such as the `AllReduced` operation of SageMaker distributed data parallel and [Horovod operations](#), is not available.

## Considerations for Monitoring System Bottlenecks and Profiling Framework Operations

- For Amazon TensorFlow, data loader metrics cannot be collected using the default `local_path` setting of the `FrameworkProfile` class. The path has to be manually configured and end in `"/"`. For example:

```
FrameworkProfile(local_path="/opt/ml/output/profiler/")
```

- For Amazon TensorFlow, the data loader profiling configuration cannot be updated while a training job is running.
- For Amazon TensorFlow, a `NoneType` error might occur when you use analysis tools and notebook examples with TensorFlow 2.3 training jobs and the detailed profiling option.
- Python profiling and detailed profiling are only supported for Keras API.
- To access the deep profiling feature for TensorFlow and PyTorch, currently you must specify the latest Amazon deep learning container images with CUDA 11. For example, you must specify the specific image URI in the TensorFlow and PyTorch estimator as follows:

- For TensorFlow

```
image_uri = f"763104351884.dkr.ecr.{region}.amazonaws.com/tensorflow-training:2.3.1-gpu-py37-cu110-ubuntu18.04"
```

- For PyTorch

```
image_uri = f"763104351884.dkr.ecr.{region}.amazonaws.com/pytorch-training:1.6.0-gpu-py36-cu110-ubuntu18.04"
```

## Considerations for Debugging Model Output Tensors

- Avoid using functional API operations. Debugger cannot collect model output tensors from PyTorch and MXNet training scripts composed of functional API operations.
- Debugger cannot collect model output tensors from the `torch.nn.functional` API operations. When you write a PyTorch training script, it is recommended to use the `torch.nn` modules instead.
- Debugger cannot collect model output tensors from MXNet functional objects in hybrid blocks. For example, the ReLU activation (`F.relu`) outputs cannot be collected from the following example of `mxnet.gluon.HybridBlock` with `F` in the `hybrid_forward` function.

```
import mxnet as mx
from mxnet.gluon import HybridBlock, nn

class Model(HybridBlock):
    def __init__(self, **kwargs):
        super(Model, self).__init__(**kwargs)
        # use name_scope to give child Blocks appropriate names.
        with self.name_scope():
            self.dense0 = nn.Dense(20)
            self.dense1 = nn.Dense(20)

    def hybrid_forward(self, F, x):
        x = F.relu(self.dense0(x))
        return F.relu(self.dense1(x))

model = Model()
model.initialize(ctx=mx.cpu(0))
model.hybridize()
model(mx.nd.zeros((10, 10), ctx=mx.cpu(0)))
```

## Amazon SageMaker Debugger Usage Statistics

Consider the following when using autogenerated reports by Amazon SageMaker Debugger.

### Debugger Profiling Report Usage

For all SageMaker training jobs, Amazon SageMaker Debugger runs the [ProfilerReport \(p. 1628\)](#) rule and autogenerates a [SageMaker Debugger Profiling Report \(p. 1701\)](#). The `ProfilerReport` rule provides a Jupyter notebook file (`profiler-report.ipynb`) that generates a corresponding HTML file (`profiler-report.html`).

Debugger collects profiling report usage statistics by including code in the Jupyter notebook that collects the unique `ProfilerReport` rule's processing job ARN if the user opens the final `profiler-report.html` file.

Debugger only collects information about whether a user opens the final HTML report. It **DOES NOT** collect any information from training jobs, training data, training scripts, processing jobs, logs, or the content of the profiling report itself.

You can opt out of the collection of usage statistics using either of the following options.

#### (Recommended) Option 1: Opt Out before Running a Training Job

To opt out, you need to add the following Debugger `ProfilerReport` rule configuration to your training job request.

SageMaker Python SDK

```
estimator=sagemaker.estimator.Estimator(
```

```
    ...
    rules=ProfilerRule.sagemaker(
        base_config=rule_configs.ProfilerReport()
        rule_parameters={"opt_out_telemetry": "True"}
    )
}
```

### Amazon CLI

```
"ProfilerRuleConfigurations": [
    {
        "RuleConfigurationName": "ProfilerReport-1234567890",
        "RuleEvaluatorImage": "895741380848.dkr.ecr.us-west-2.amazonaws.com/sagemaker-
debugger-rules:latest",
        "RuleParameters": {
            "rule_to_invoke": "ProfilerReport",
            "opt_out_telemetry": "True"
        }
    }
]
```

### Amazon SDK for Python (Boto3)

```
ProfilerRuleConfigurations=[
    {
        'RuleConfigurationName': 'ProfilerReport-1234567890',
        'RuleEvaluatorImage': '895741380848.dkr.ecr.us-west-2.amazonaws.com/sagemaker-
debugger-rules:latest',
        'RuleParameters': {
            'rule_to_invoke': 'ProfilerReport',
            'opt_out_telemetry': 'True'
        }
    }
]
```

## Option 2: Opt Out after a Training Job Has Completed

To opt out after training has completed, you need to modify the `profiler-report.ipynb` file.

### Note

HTML reports autogenerated without **Option 1** already added to your training job request still report the usage statistics even after you opt out using **Option 2**.

1. Follow the instructions on downloading the Debugger profiling report files in the [Download a Debugger Profiling Report \(p. 1701\)](#) page.
2. In the `/ProfilerReport-1234567890/profiler-output` directory, open `profiler-report.ipynb`.
3. Add `opt_out=True` to the `setup_profiler_report()` function in the fifth code cell as shown in the following example code:

```
setup_profiler_report(processing_job_arn, opt_out=True)
```

4. Run the code cell to finish opting out.

# Perform Automatic Model Tuning with SageMaker

Amazon SageMaker automatic model tuning, also known as hyperparameter tuning, finds the best version of a model by running many training jobs on your dataset using the algorithm and ranges of hyperparameters that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by a metric that you choose.

For example, suppose that you want to solve a [binary classification](#) problem on a marketing dataset. Your goal is to maximize the [area under the curve \(auc\)](#) metric of the algorithm by training an [XGBoost Algorithm \(p. 1524\)](#) model. You don't know which values of the `eta`, `alpha`, `min_child_weight`, and `max_depth` hyperparameters to use to train the best model. To find the best values for these hyperparameters, you can specify ranges of values that SageMaker hyperparameter tuning searches to find the combination of values that results in the training job that performs the best as measured by the objective metric that you chose. Hyperparameter tuning launches training jobs that use hyperparameter values in the ranges that you specified, and returns the training job with highest auc.

You can use SageMaker automatic model tuning with built-in algorithms, custom algorithms, and SageMaker pre-built containers for machine learning frameworks.

Amazon SageMaker automatic model tuning can use Amazon EC2 Spot instance to optimize costs when running training jobs. For more information on managed spot training, see [Managed Spot Training in Amazon SageMaker \(p. 1859\)](#).

Before you start using hyperparameter tuning, you should have a well-defined machine learning problem, including the following:

- A dataset
- An understanding of the type of algorithm you need to train
- A clear understanding of how you measure success

You should also prepare your dataset and algorithm so that they work in SageMaker and successfully run a training job at least once. For information about setting up and running a training job, see [Get Started with Amazon SageMaker \(p. 34\)](#).

## Topics

- [How Hyperparameter Tuning Works \(p. 1745\)](#)
- [Define Metrics \(p. 1747\)](#)
- [Define Hyperparameter Ranges \(p. 1748\)](#)
- [Tune Multiple Algorithms with Hyperparameter Optimization to Find the Best Model \(p. 1749\)](#)
- [Example: Hyperparameter Tuning Job \(p. 1754\)](#)
- [Stop Training Jobs Early \(p. 1764\)](#)
- [Run a Warm Start Hyperparameter Tuning Job \(p. 1765\)](#)
- [Resource Limits for Automatic Model Tuning \(p. 1769\)](#)
- [Best Practices for Hyperparameter Tuning \(p. 1770\)](#)

## How Hyperparameter Tuning Works

### Random Search

In a random search, hyperparameter tuning chooses a random combination of values from within the ranges that you specify for hyperparameters for each training job it launches. Because the choice of hyperparameter values doesn't depend on the results of previous training jobs, you can run the maximum number of concurrent training jobs without affecting the performance of the search.

For an example notebook that uses random search, see [https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter\\_tuning/xgboost\\_random\\_log/hpo\\_xgboost\\_random\\_log.ipynb](https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/xgboost_random_log/hpo_xgboost_random_log.ipynb).

## Bayesian Search

Bayesian search treats hyperparameter tuning like a [\[regression\]](#) problem. Given a set of input features (the hyperparameters), hyperparameter tuning optimizes a model for the metric that you choose. To solve a regression problem, hyperparameter tuning makes guesses about which hyperparameter combinations are likely to get the best results, and runs training jobs to test these values. After testing the first set of hyperparameter values, hyperparameter tuning uses regression to choose the next set of hyperparameter values to test.

Hyperparameter tuning uses a Amazon SageMaker implementation of Bayesian optimization.

When choosing the best hyperparameters for the next training job, hyperparameter tuning considers everything that it knows about this problem so far. Sometimes it chooses a combination of hyperparameter values close to the combination that resulted in the best previous training job to incrementally improve performance. This allows hyperparameter tuning to exploit the best known results. Other times, it chooses a set of hyperparameter values far removed from those it has tried. This allows it to explore the range of hyperparameter values to try to find new areas that are not well understood. The explore/exploit trade-off is common in many machine learning problems.

For more information about Bayesian optimization, see the following:

### Basic Topics on Bayesian Optimization

- [A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning](#)
- [Practical Bayesian Optimization of Machine Learning Algorithms](#)
- [Taking the Human Out of the Loop: A Review of Bayesian Optimization](#)

### Speeding up Bayesian Optimization

- [Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization](#)
- [Google Vizier: A Service for Black-Box Optimization](#)
- [Learning Curve Prediction with Bayesian Neural Networks](#)
- [Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves](#)

### Advanced Modeling and Transfer Learning

- [Scalable Hyperparameter Transfer Learning](#)
- [Bayesian Optimization with Tree-structured Dependencies](#)
- [Bayesian Optimization with Robust Bayesian Neural Networks](#)
- [Scalable Bayesian Optimization Using Deep Neural Networks](#)
- [Input Warping for Bayesian Optimization of Non-stationary Functions](#)

#### Note

Hyperparameter tuning might not improve your model. It is an advanced tool for building machine solutions, and, as such, should be considered part of the scientific development process.

When you build complex machine learning systems like deep learning neural networks, exploring all of the possible combinations is impractical. Hyperparameter tuning can accelerate

your productivity by trying many variations of a model, focusing on the most promising combinations of hyperparameter values within the ranges that you specify. To get good results, you need to choose the right ranges to explore. Because the algorithm itself is stochastic, it's possible that the hyperparameter tuning model will fail to converge on the best answer, even if the best possible combination of values is within the ranges that you choose.

## Define Metrics

When you use one of the Amazon SageMaker built-in algorithms, you don't need to define metrics. Built-in algorithms automatically send metrics to hyperparameter tuning. You do need to choose one of the metrics that the built-in algorithm emits as the objective metric for the tuning job. For a list of metrics that a built-in algorithm emits, see the *Metrics* table for the algorithm in [Use Amazon SageMaker Built-in Algorithms \(p. 718\)](#).

To optimize hyperparameters for a machine learning model, a tuning job evaluates the training jobs it launches by using a metric that the training algorithm writes to logs. Amazon SageMaker hyperparameter tuning parses your algorithm's `stdout` and `stderr` streams to find algorithm metrics, such as loss or validation-accuracy, that show how well the model is performing on the dataset.

**Note**

These are the same metrics that SageMaker sends to CloudWatch Logs. For more information, see [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 2534\)](#).

If you use your own algorithm for hyperparameter tuning, make sure that your algorithm emits at least one metric by writing evaluation data to `stderr` or `stdout`.

**Note**

Hyperparameter tuning sends an additional hyperparameter, `_tuning_objective_metric` to the training algorithm. This hyperparameter specifies the objective metric being used for the hyperparameter tuning job, so that your algorithm can use that information during training.

You can define up to 20 metrics for your tuning job to monitor. You choose one of those metrics to be the objective metric, which hyperparameter tuning uses to evaluate the training jobs. The hyperparameter tuning job returns the training job that returned the best value for the objective metric as the best training job.

You define metrics for a tuning job by specifying a name and a regular expression for each metric that your tuning job monitors. Design the regular expressions to capture the values of metrics that your algorithm emits. You pass these metrics to the `CreateHyperParameterTuningJob` operation in the `TrainingJobDefinition` parameter as the `MetricDefinitions` field of the `AlgorithmSpecification` field.

The following example defines 4 metrics:

```
=[
  {
    "Name": "loss",
    "Regex": "Loss = (.*) ;",
  },
  {
    "Name": "ganloss",
    "Regex": "GAN_loss=(.*?);",
  },
  {
    "Name": "discloss",
    "Regex": "disc_train_loss=(.*?);",
  },
  {
    "Name": "disc-combined",
    "Regex": "disc-combined=(.*?);",
  },
]
```

]

The following is an example of the log that the algorithm writes:

```
GAN_loss=0.138318; Scaled_reg=2.654134; disc:[-0.017371,0.102429] real 93.3% gen 0.0%
disc-combined=0.000000; disc_train_loss=1.374587; Loss = 16.020744; Iteration 0 took
0.704s; Elapsed=0s
```

Use the regular expression (regex) to match the algorithm's log output and capture the numeric values of metrics. For example, in the regex for the loss metric defined above, the first part of the regex finds the exact text "Loss = ", and the expression `(.*?)`; captures zero or more of any character until the first semicolon character. In this expression, the parenthesis tell the regex to capture what is inside them, `.` means any character, `*` means zero or more, and `?` means capture only until the first instance of the `;` character.

Choose one of the metrics that you define as the objective metric for the tuning job. If you are using the API, specify the value of the `name` key in the `HyperParameterTuningJobObjective` field of the `HyperParameterTuningJobConfig` parameter that you send to the [CreateHyperParameterTuningJob](#) operation.

## Define Hyperparameter Ranges

Hyperparameter tuning finds the best hyperparameter values for your model by searching over a set of values that you specify for each of the hyperparameters that are tunable. Choosing hyperparameters and ranges significantly affects the performance of your tuning job. For guidance on choosing hyperparameters and ranges, see [Best Practices for Hyperparameter Tuning \(p. 1770\)](#).

To define hyperparameter ranges by using the low-level API, you specify the names of hyperparameters and ranges of values in the `ParameterRanges` field of the `HyperParameterTuningJobConfig` parameter that you pass to the [CreateHyperParameterTuningJob](#) operation. The `ParameterRanges` field has three subfields, one for each of the categorical, integer, and continuous hyperparameter ranges. You can define up to 20 hyperparameters to search over. Each value of a categorical hyperparameter range counts as a hyperparameter against the limit. Hyperparameter ranges have the following structure:

```
"ParameterRanges": {
    "CategoricalParameterRanges": [
        {
            "Name": "tree_method",
            "Values": ["auto", "exact", "approx", "hist"]
        }
    ],
    "ContinuousParameterRanges": [
        {
            "Name": "eta",
            "MaxValue": "0.5",
            "MinValue": "0",
            "ScalingType": "Auto"
        }
    ],
    "IntegerParameterRanges": [
        {
            "Name": "max_depth",
            "MaxValue": "10",
            "MinValue": "1",
            "ScalingType": "Auto"
        }
    ]
}
```

## Hyperparameter Scaling

For integer and continuous hyperparameter ranges, you can choose the scale you want hyperparameter tuning to use to search the range of values by specifying a value for the `ScalingType` field of the hyperparameter range. You can choose from the following scaling types:

### Auto

SageMaker hyperparameter tuning chooses the best scale for the hyperparameter.

### Linear

Hyperparameter tuning searches the values in the hyperparameter range by using a linear scale. Typically, you choose this if the range of all values from the lowest to the highest is relatively small (within one order of magnitude), because uniformly searching values from the range will give you a reasonable exploration of the entire range.

### Logarithmic

Hyperparameter tuning searches the values in the hyperparameter range by using a logarithmic scale.

Logarithmic scaling works only for ranges that have only values greater than 0.

Choose logarithmic scaling when you are searching a range that spans several orders of magnitude. For example, if you are tuning a [Tune a linear learner model \(p. 1442\)](#) model, and you specify a range of values between .0001 and 1.0 for the `learning_rate` hyperparameter, searching uniformly on a logarithmic scale gives you a better sample of the entire range than searching on a linear scale would, because searching on a linear scale would, on average, devote 90 percent of your training budget to only the values between .1 and 1.0, leaving only 10 percent of your training budget for the values between .0001 and .1.

### ReverseLogarithmic

Hyperparameter tuning searches the values in the hyperparameter range by using a reverse logarithmic scale. reverse logarithmic scaling is supported only for continuous hyperparameter ranges. It is not supported for integer hyperparameter ranges.

Reverse logarithmic scaling works only for ranges that are entirely within the range  $0 \leq x < 1.0$ .

Choose reverse logarithmic scaling when you are searching a range that is highly sensitive to small changes that are very close to 1.

For an example notebook that uses hyperparameter scaling, see [https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter\\_tuning/xgboost\\_random\\_log/hpo\\_xgboost\\_random\\_log.ipynb](https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/xgboost_random_log/hpo_xgboost_random_log.ipynb).

## Tune Multiple Algorithms with Hyperparameter Optimization to Find the Best Model

To create a new hyperparameter optimization (HPO) job with Amazon SageMaker that tunes multiple algorithms, you must provide job settings that apply to all of the algorithms to be tested and a training definition for each of these algorithms. You must also specify the resources you want to use for the tuning job.

- The **job settings** to configure include warm starting, early stopping, and the tuning strategy. Warm starting and early stopping are available only when tuning a single algorithm.
- The **training job definition** to specify the name, algorithm source, objective metric, and the range of values, when required, to configure the set of hyperparameter values for each training job. It

configures the channels for data inputs, data output locations, and any checkpoint storage locations for each training job. The definition also configures the resources to deploy for each training job, including instance types and counts, managed spot training, and stopping conditions.

- The **tuning job resources**: to deploy, including the maximum number of concurrent training jobs that a hyperparameter tuning job can run concurrently and the maximum number of training jobs that the hyperparameter tuning job can run.

## Get Started

You can create a new hyperparameter tuning job, clone a job, add or edit tags to a job from the console. You can also use the search feature to find jobs by their name, creation time, or status. Alternatively, you can also hyperparameter tuning jobs with the SageMaker API.

- **In the console:** To create a new job, open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>, choose **Hyperparameter tuning jobs** from the **Training**, menu, and then choose **Create hyperparameter tuning job**. Then following the configuration steps to create a training job for each algorithm that you want to use. These steps are documented in the [Create a Hyperparameter Optimization Tuning Job for One or More Algorithms \(Console\) \(p. 1750\)](#) topic.

### Note

When you start the configuration steps, note that the warm start and early stopping features are not available to use with multi-algorithm HPO. If you want to use these features, you can only tune a single algorithm at a time.

- **With the API:** For instructions on using the SageMaker API to create a hyperparameter tuning job, see [Example: Hyperparameter Tuning Job](#). When you call `CreateHyperParameterTuningJob` to tune multiple algorithms, you must provide a list of training definitions using `TrainingJobDefinitions` instead of specifying a single `TrainingJobDefinition`. You must choose just one of these definition types depending on the number of algorithms being tuned.

### Topics

- [Create a Hyperparameter Optimization Tuning Job for One or More Algorithms \(Console\) \(p. 1750\)](#)
- [Manage Hyperparameter Tuning and Training Jobs \(p. 1754\)](#)

## Create a Hyperparameter Optimization Tuning Job for One or More Algorithms (Console)

To create a new hyperparameter optimization (HPO) tuning job for one or more algorithms, you need to define the settings for the tuning job, create training job definitions for each algorithm being tuned, and configure the resources for the tuning job.

### Topics

- [Define job settings \(p. 1750\)](#)
- [Create Training Job Definitions \(p. 1751\)](#)
- [Configure Tuning Job Resources \(p. 1753\)](#)
- [Review and Create HPO Tuning Job \(p. 1753\)](#)

## Define job settings

Your tuning job settings are applied across all of the algorithms in the HPO tuning job. Warm start and early stopping are available only when tuning a single algorithm. After you define the job settings you will create individual training definitions for each algorithm or variation you want to tune.

## Warm Start

If you cloned this job, you can choose to use the results from a previous tuning job to improve the performance of this new tuning job. This is the warm start feature and it is only available when tuning a single algorithm. When you choose this option, you can choose up to five previous hyperparameter tuning jobs to use. Alternatively, you can use transfer learning to add additional data to the parent tuning job. When you select this option, you choose one previous tuning job as the parent.

### Note

Warm start is compatible only with tuning jobs created after October 1, 2018. For more information, see [Run a warm start job](#).

## Early Stopping

To reduce compute time and avoid overfitting your model, training jobs can be stopped early when they are unlikely to improve the current best objective metric of the hyperparameter tuning job.. Like warm start, this feature is only available when tuning a single algorithm. This is an automatic feature without configuration options, and it's disabled by default. For more information on how early stopping works, the algorithms that support it, and how to use it with your own algorithms, see [Stop Training Jobs Early](#).

## Tuning Strategy

Tuning strategy can be either random or Bayesian. It specifies how the automatic tuning searches over specified hyperparameter ranges. You specify the ranges in a later step. Random search chooses random combinations of values from the specified ranges and can be run concurrently. Bayesian search chooses values based on what is likely to get the best result given what is known about the history of previous selections. For more information search strategies, see [How Hyperparameter Tuning Works](#).

## Tags

You enter tags as key-value pairs to assign metadata to tuning jobs to help you manage them. Values are not required. You can use just the key. To see the keys associated with a job, choose the **Tags** tab on the details page for tuning job. For more information about using tags for tuning jobs, see [Manage Hyperparameter Tuning and Training Jobs \(p. 1754\)](#)

## Create Training Job Definitions

To create a training job definition, you need to configure the algorithm and parameters, define the data input and output, and configure resources. You must provide at least one [TrainingJobDefinition](#) for each HPO tuning job. Each training definition specifies the configuration for an algorithm. To create several definitions for your training job you can clone a job definition. Cloning a job can save time as it copies all of the job settings, including data channels, S3 storage locations for output artifacts. You can then edit the cloned job just for changes needed to configure the algorithm options.

### Topics

- [Configure algorithm and parameters \(p. 1751\)](#)
- [Define Data Input and Output \(p. 1752\)](#)
- [Configure Training Job Resources \(p. 1753\)](#)
- [Add or Clone a Training Job \(p. 1753\)](#)

## Configure algorithm and parameters

Each training job definition for a tuning job requires a name, permission to access services, and the specification of algorithm options, an objective metric, and the range of values, when required, to configure the set of hyperparameter values for each training job.

### Name

Provide a unique name for the training definition.

### Permissions

Amazon SageMaker requires permissions to call other services on your behalf. Choose an IAM role or let Amazon create a role that has the `AmazonSageMakerFullAccess` IAM policy attached.

### Optional Security Settings

The network isolation setting prevents the container from making any outbound network calls. This is required for Amazon Marketplace machine learning offerings.

You can also choose to use a private VPC.

#### Note

Inter-container encryption is only available when creating job definitions from the API.

### Algorithm Options

You can choose one of the built-in algorithms, your own algorithm, your own container with an algorithm, or you can subscribe to an algorithm from Amazon Marketplace.

- If you choose a built-in algorithm, it has the ECR image information prepopulated.
- If you choose your own container, you must specify the ECR image information. You can select the input mode for the algorithm as file or pipe.
- If you plan to supply your data using a .CSV file from Amazon S3, you should select the file.

### Metrics

When you choose a built-in algorithm, metrics are provided for you. If you choose your own algorithm, you need to define your metrics. You can define up to 20 metrics for your tuning job to monitor, one of which must be chosen as the objective metric. For more information on how to define a metric for a tuning job, see [Define Metrics \(p. 1747\)](#).

### Objective Metric

To find the best training job, set an objective metric and whether to maximize or minimize it. After the training job is complete, you can view the tuning job detail page for a summary of the best training job found using this objective metric.

### Hyperparameter Configuration

When you choose a built-in algorithm, the default values for its hyperparameters are set for you, using ranges that are optimized for the algorithm being tuned. You can change these values as you see fit. For example, instead of a range, you can set a fixed value for a hyperparameter by setting the parameter's type to `static`. Each algorithm has different required and optional parameters. For more information, see [Best Practices for Hyperparameter Tuning](#) and [Define Hyperparameter Ranges](#).

### Define Data Input and Output

Each training job definition for a tuning job must configures the channels for data inputs, data output locations, and optionally any checkpoint storage locations for each training job.

### Input Data Configuration

Input data is defined by channels, each with their own source location (Amazon S3 or Amazon Elastic File System), compression, and format options. You can define up to 20 channels of input sources. If the

algorithm you chose supports multiple input channels, you can specify those too. For example, when using the [XGBoost churn prediction notebook](#), you could add two channels: train and validation.

### Checkpoint Configuration

Checkpoints are periodically generated during training. You must choose an Amazon S3 location for the checkpoints to be saved. Checkpoints are used in metrics reporting, and are also used to resume managed spot training jobs. For more information, see [Use Checkpoints in Amazon SageMaker \(p. 1860\)](#).

### Output Data Configuration

You must define an Amazon S3 location for the artifacts of the training job to be stored. You have the option of adding encryption to the output using an Amazon Key Management Service (Amazon KMS) key.

### Configure Training Job Resources

Each training job definition for a tuning job must configure the resources to deploy, including instance types and counts, managed spot training, and stopping conditions.

#### Resource Configuration

Each training definition can have a different resource configuration. You choose the instance type and number of nodes.

#### Managed spot training

You can save computer costs for jobs if you have flexibility in start and end times by allowing SageMaker to use spare capacity to run jobs. For more information, see [Managed Spot Training in Amazon SageMaker \(p. 1859\)](#).

#### Stopping condition

The stopping condition specifies the maximum duration allowed per training job.

### Add or Clone a Training Job

Once you have created a training job definition for a tuning job, you are returned to the **Training Job Definition(s)** panel where you can create additional training job definitions to train additional algorithms. You can select the **Add training job definition** and work through the steps to define a training job again or choose **Clone** from the **Action** menu to replicate an existing training job definition and edit it for the new algorithm. The clone option can save time as it copies all of the job's settings, including the data channels, S3 storage locations. For more information on cloning, see [Manage Hyperparameter Tuning and Training Jobs \(p. 1754\)](#)

### Configure Tuning Job Resources

#### Resource Limits

You can specify the maximum number of concurrent training jobs that a hyperparameter tuning job can run concurrently (10 at most) and the maximum number of training jobs that the hyperparameter tuning job can run (500 at most). The number of parallel jobs should not exceed the number of nodes you have requested across all of your training definitions. The total number of jobs can't exceed the number of jobs that your definitions are expected to run.

### Review and Create HPO Tuning Job

Review the job settings, the training job definition(s), and resource limits. Then select **Create hyperparameter tuning job**.

## Manage Hyperparameter Tuning and Training Jobs

A tuning job can contain many training jobs and creating and managing these jobs and their definitions can become a complex and onerous task. SageMaker provides tools to help facilitate the management of these jobs. Tuning jobs you have run can be accessed from the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>. Select **Hyperparameter tuning job** from the **Training** menu to see the list. This page is also where you start the procedure to create a new tuning job by selecting **Create hyperparameter tuning job**.

To see the training jobs run as part of a tuning job, select one of the hyperparameter tuning jobs from the list. The tabs on the tuning job page allow you to inspect the training jobs, their definitions, the tags and configuration used for the tuning job, and the best training job found during tuning. You can select the best training job or any of the other training jobs that belong to the tuning job to see all of their settings. From here you can create a model that uses the hyperparameter values found by a training job by selecting **Create Model** or you can clone the training job by selecting **Clone**.

### Cloning

You can save time by cloning a training job that belongs to a hyperparameter tuning job. Cloning copies all of the job's settings, including data channels, S3 storage locations for output artifacts. You can do this for training jobs you have already run from the the tuning job page, as just described, or when you are creating additional training job definitions while creating a hyperparameter tuning job, as described in [Add or Clone a Training Job \(p. 1753\)](#) step of that procedure.

### Tagging

Automatic Model Tuning launches multiple training jobs within a single parent tuning job to discover the ideal weighting of model hyperparameters. Tags can be added to the parent tuning job as described in the [Define job settings \(p. 1750\)](#) section and these tags are then propagated to the individual training jobs underneath. Customers can use these tags for purposes such as cost allocation or access control. To add tags using the SageMaker SDK, use [AddTags](#) API. For more information about using tagging for Amazon resources, see [Tagging Amazon resources](#).

## Example: Hyperparameter Tuning Job

This example shows how to create a new notebook for configuring and launching a hyperparameter tuning job. The tuning job uses the [XGBoost Algorithm \(p. 1524\)](#) to train a model to predict whether a customer will enroll for a term deposit at a bank after being contacted by phone.

You use the low-level Amazon SDK for Python (Boto) to configure and launch the hyperparameter tuning job, and the Amazon Web Services Management Console to monitor the status of hyperparameter tuning jobs. You can also use the Amazon SageMaker high-level [Amazon SageMaker Python SDK](#) to configure, run, monitor, and analyze hyperparameter tuning jobs. For more information, see <https://github.com/aws/sagemaker-python-sdk>.

## Prerequisites

To run the code in this example, you need

- [An Amazon account and an administrator user \(p. 34\)](#)
- An Amazon S3 bucket for storing your training dataset and the model artifacts created during training
- [A running SageMaker notebook instance \(p. 53\)](#)

### Topics

- [Create a Notebook \(p. 1755\)](#)

- [Get the Amazon SageMaker Boto 3 Client \(p. 1755\)](#)
- [Get the SageMaker Execution Role \(p. 1756\)](#)
- [Specify a S3 Bucket to Upload Training Datasets and Store Output Data \(p. 1756\)](#)
- [Download, Prepare, and Upload Training Data \(p. 1756\)](#)
- [Configure and Launch a Hyperparameter Tuning Job \(p. 1757\)](#)
- [Monitor the Progress of a Hyperparameter Tuning Job \(p. 1761\)](#)
- [Clean up \(p. 1763\)](#)

## Create a Notebook

Create a Jupyter notebook that contains a pre-installed environment with the default Anaconda installation and Python3.

### To create a Jupyter notebook

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Open a running notebook instance, by choosing **Open** next to its name. The Jupyter notebook server page appears:



3. To create a notebook, choose **Files**, **New**, and **conda\_python3**..
4. Name the notebook.

## Next Step

[Get the Amazon SageMaker Boto 3 Client \(p. 1755\)](#)

## Get the Amazon SageMaker Boto 3 Client

Import Amazon SageMaker Python SDK, Amazon SDK for Python (Boto3), and other Python libraries. In a new Jupyter notebook, paste the following code to the first cell:

```
import sagemaker
import boto3

import numpy as np                               # For performing matrix operations and
numerical processing
import pandas as pd                             # For manipulating tabular data
from time import gmtime, strftime
import os

region = boto3.Session().region_name
smclient = boto3.Session().client('sagemaker')
```

The preceding code cell defines `region` and `smclient` objects that you will use to call the built-in XGBoost algorithm and set the SageMaker hyperparameter tuning job.

## Next Step

[Get the SageMaker Execution Role \(p. 1756\)](#)

## Get the SageMaker Execution Role

Get the execution role for the notebook instance. This is the IAM role that you created for your notebook instance. You pass the role to the tuning job.

```
from sagemaker import get_execution_role  
  
role = get_execution_role()  
print(role)
```

## Next Step

[Specify a S3 Bucket to Upload Training Datasets and Store Output Data \(p. 1756\)](#)

## Specify a S3 Bucket to Upload Training Datasets and Store Output Data

Set up a S3 bucket to upload training datasets and save training output data.

### To use a default S3 bucket

Use the following code to specify the default S3 bucket allocated for your SageMaker session. `prefix` is the path within the bucket where SageMaker stores the data for the current training job.

```
sess = sagemaker.Session()  
bucket = sess.default_bucket() # Set a default S3 bucket  
prefix = 'DEMO-automatic-model-tuning-xgboost-dm'
```

### (Optional) To use a specific S3 bucket

If you want to use a specific S3 bucket, use the following code and replace the strings to the exact name of the S3 bucket. The name of the bucket must contain `sagemaker`, and be globally unique. The bucket must be in the same Amazon Region as the notebook instance that you use for this example.

```
bucket = "sagemaker-your-preferred-s3-bucket"
```

### Note

The name of the bucket doesn't need to contain `sagemaker` if the IAM role that you use to run the hyperparameter tuning job has a policy that gives the `S3FullAccess` permission.

## Next Step

[Download, Prepare, and Upload Training Data \(p. 1756\)](#)

## Download, Prepare, and Upload Training Data

For this example, you use a training dataset of information about bank customers that includes the customer's job, marital status, and how they were contacted during the bank's direct marketing campaign. To use a dataset for a hyperparameter tuning job, you download it, transform the data, and then upload it to an Amazon S3 bucket.

For more information about the dataset and the data transformation that the example performs, see the `hpo_xgboost_direct_marketing_sagemaker_APIs` notebook in the **Hyperparameter Tuning** section of the **SageMaker Examples** tab in your notebook instance.

## Download and Explore the Training Dataset

To download and explore the dataset, run the following code in your notebook:

```
!wget -N https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank-additional.zip
!unzip -o bank-additional.zip
data = pd.read_csv('./bank-additional/bank-additional-full.csv', sep=';')
pd.set_option('display.max_columns', 500)      # Make sure we can see all of the columns
pd.set_option('display.max_rows', 5)            # Keep the output on one page
data
```

## Prepare and Upload Data

Before creating the hyperparameter tuning job, prepare the data and upload it to an S3 bucket where the hyperparameter tuning job can access it.

Run the following code in your notebook:

```
data['no_previous_contact'] = np.where(data['pdays'] == 999, 1, 0)
    # Indicator variable to capture when pdays takes a value of 999
data['not_working'] = np.where(np.in1d(data['job'], ['student', 'retired', 'unemployed']), 1, 0)
    # Indicator for individuals not actively employed
model_data = pd.get_dummies(data)
    # Convert categorical variables to sets of indicators
model_data = model_data.drop(['duration', 'emp.var.rate', 'cons.price.idx',
    'cons.conf.idx', 'euribor3m', 'nr.employed'], axis=1)

train_data, validation_data, test_data = np.split(model_data.sample(frac=1,
    random_state=1729), [int(0.7 * len(model_data)), int(0.9*len(model_data))])

pd.concat([train_data['y_yes'], train_data.drop(['y_no', 'y_yes'], axis=1)],
    axis=1).to_csv('train.csv', index=False, header=False)
pd.concat([validation_data['y_yes'], validation_data.drop(['y_no', 'y_yes'], axis=1)],
    axis=1).to_csv('validation.csv', index=False, header=False)
pd.concat([test_data['y_yes'], test_data.drop(['y_no', 'y_yes'], axis=1)],
    axis=1).to_csv('test.csv', index=False, header=False)

boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train/train.csv')).upload_file('train.csv')
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'validation/validation.csv')).upload_file('validation.csv')
```

## Next Step

[Configure and Launch a Hyperparameter Tuning Job \(p. 1757\)](#)

## Configure and Launch a Hyperparameter Tuning Job

To configure and launch a hyperparameter tuning job, complete the following steps.

### Topics

- [Specify the Hyperparameter Tuning Job Settings \(p. 1758\)](#)
- [Configure the Training Jobs \(p. 1759\)](#)

- [Name and Launch the Hyperparameter Tuning Job \(p. 1761\)](#)
- [Next Step \(p. 1761\)](#)

## Specify the Hyperparameter Tuning Job Settings

To specify settings for the hyperparameter tuning job, you define a JSON object. You pass the object as the value of the `HyperParameterTuningJobConfig` parameter to [CreateHyperParameterTuningJob](#) when you create the tuning job.

In this JSON object, you specify:

- The ranges of hyperparameters that you want to tune. For more information, see [Define Hyperparameter Ranges \(p. 1748\)](#)
- The limits of the resource that the hyperparameter tuning job can consume.
- The objective metric for the hyperparameter tuning job. An *objective metric* is the metric that the hyperparameter tuning job uses to evaluate the training job that it launches.

**Note**

To use your own algorithm for hyperparameter tuning, you need to define metrics for your algorithm. For information, see [Define Metrics \(p. 1747\)](#).

The hyperparameter tuning job defines ranges for the `eta`, `alpha`, `min_child_weight`, and `max_depth` hyperparameters of the [XGBoost Algorithm \(p. 1524\)](#) built-in algorithm. The objective metric for the hyperparameter tuning job maximizes the `validation:auc` metric that the algorithm sends to CloudWatch Logs.

```
tuning_job_config = {
    "ParameterRanges": {
        "CategoricalParameterRanges": [],
        "ContinuousParameterRanges": [
            {
                "MaxValue": "1",
                "MinValue": "0",
                "Name": "eta"
            },
            {
                "MaxValue": "2",
                "MinValue": "0",
                "Name": "alpha"
            },
            {
                "MaxValue": "10",
                "MinValue": "1",
                "Name": "min_child_weight"
            }
        ],
        "IntegerParameterRanges": [
            {
                "MaxValue": "10",
                "MinValue": "1",
                "Name": "max_depth"
            }
        ]
    },
    "ResourceLimits": {
        "MaxNumberOfTrainingJobs": 20,
        "MaxParallelTrainingJobs": 3
    },
    "Strategy": "Bayesian",
    "HyperParameterTuningJobObjective": {
```

```

        "MetricName": "validation:auc",
        "Type": "Maximize"
    }
}
```

## Configure the Training Jobs

To configure the training jobs that the tuning job launches, define a JSON object that you pass as the value of the `TrainingJobDefinition` parameter of the [CreateHyperParameterTuningJob](#) call.

In this JSON object, you specify:

- Optional—Metrics that the training jobs emit.

**Note**

Define metrics only when you use a custom training algorithm. Because this example uses a built-in algorithm, you don't specify metrics. For information about defining metrics, see [Define Metrics \(p. 1747\)](#).

- The container image that specifies the training algorithm.
- The input configuration for your training and test data.
- The storage location for the algorithm's output. Specify the S3 bucket where you want to store the output of the training jobs.
- The values of algorithm hyperparameters that are not tuned in the tuning job.
- The type of instance to use for the training jobs.
- The stopping condition for the training jobs. This is the maximum duration for each training job.

In this example, we set static values for the `eval_metric`, `num_round`, `objective`, `rate_drop`, and `tweedie_variance_power` parameters of the [XGBoost Algorithm \(p. 1524\)](#) built-in algorithm.

SageMaker Python SDK v1

```

from sagemaker.amazon.amazon_estimator import get_image_uri
training_image = get_image_uri(region, 'xgboost', repo_version='1.0-1')

s3_input_train = 's3://{}//{}//train'.format(bucket, prefix)
s3_input_validation = 's3://{}//{}//validation/'.format(bucket, prefix)

training_job_definition = {
    "AlgorithmSpecification": {
        "TrainingImage": training_image,
        "TrainingInputMode": "File"
    },
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_train
                }
            }
        },
        {
            "ChannelName": "validation",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_validation
                }
            }
        }
    ]
}
```

```

        "DataSource": {
            "S3DataSource": {
                "S3DataDistributionType": "FullyReplicated",
                "S3DataType": "S3Prefix",
                "S3Uri": s3_input_validation
            }
        }
    ],
    "OutputDataConfig": {
        "S3OutputPath": "s3://{}{}/output".format(bucket,prefix)
    },
    "ResourceConfig": {
        "InstanceCount": 2,
        "InstanceType": "ml.c4.2xlarge",
        "VolumeSizeInGB": 10
    },
    "RoleArn": role,
    "StaticHyperParameters": {
        "eval_metric": "auc",
        "num_round": "100",
        "objective": "binary:logistic",
        "rate_drop": "0.3",
        "tweedie_variance_power": "1.4"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 43200
    }
}

```

#### SageMaker Python SDK v2

```

training_image = sagemaker.image_uris.retrieve('xgboost', region, '1.0-1')

s3_input_train = 's3://{}{}/train'.format(bucket, prefix)
s3_input_validation = 's3://{}{}/validation/'.format(bucket, prefix)

training_job_definition = {
    "AlgorithmSpecification": {
        "TrainingImage": training_image,
        "TrainingInputMode": "File"
    },
    "InputDataConfig": [
        {
            "ChannelName": "train",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_train
                }
            }
        },
        {
            "ChannelName": "validation",
            "CompressionType": "None",
            "ContentType": "csv",
            "DataSource": {
                "S3DataSource": {
                    "S3DataDistributionType": "FullyReplicated",
                    "S3DataType": "S3Prefix",
                    "S3Uri": s3_input_validation
                }
            }
        }
    ]
}

```

```
        }
    ],
    "OutputDataConfig": {
        "S3OutputPath": "s3://{}{}/output".format(bucket,prefix)
    },
    "ResourceConfig": {
        "InstanceCount": 2,
        "InstanceType": "ml.c4.2xlarge",
        "VolumeSizeInGB": 10
    },
    "RoleArn": role,
    "StaticHyperParameters": {
        "eval_metric": "auc",
        "num_round": "100",
        "objective": "binary:logistic",
        "rate_drop": "0.3",
        "tweedie_variance_power": "1.4"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 43200
    }
}
```

## Name and Launch the Hyperparameter Tuning Job

Now you can provide a name for the hyperparameter tuning job and then launch it by calling the [CreateHyperParameterTuningJob](#) API. Pass `tuning_job_config`, and `training_job_definition` that you created in previous steps as the values of the parameters.

```
tuning_job_name = "MyTuningJob"
smclient.create_hyper_parameter_tuning_job(HyperParameterTuningJobName = tuning_job_name,
                                            HyperParameterTuningJobConfig =
                                            tuning_job_config,
                                            TrainingJobDefinition = training_job_definition)
```

## Next Step

[Monitor the Progress of a Hyperparameter Tuning Job \(p. 1761\)](#)

## Monitor the Progress of a Hyperparameter Tuning Job

To monitor the progress of a hyperparameter tuning job and the training jobs that it launches, use the Amazon SageMaker console.

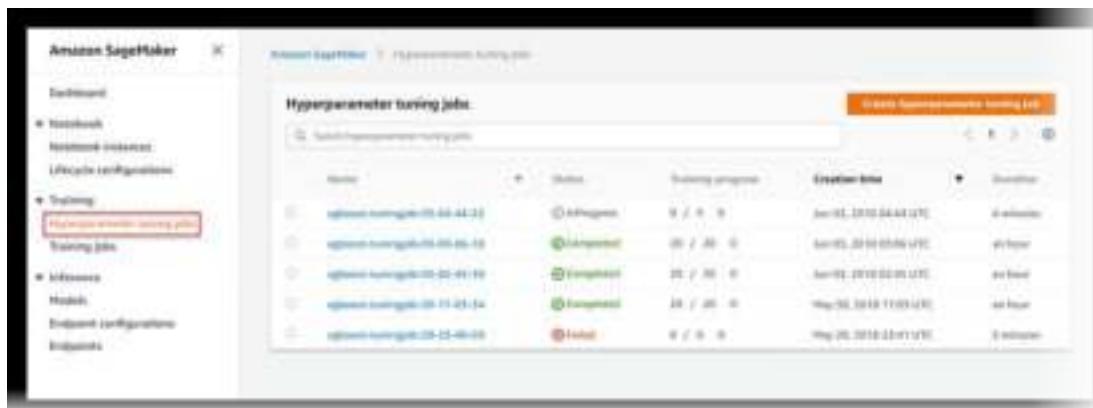
### Topics

- [View the Status of the Hyperparameter Tuning Job \(p. 1761\)](#)
- [View the Status of the Training Jobs \(p. 1762\)](#)
- [View the Best Training Job \(p. 1763\)](#)

## View the Status of the Hyperparameter Tuning Job

### To view the status of the hyperparameter tuning job

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Choose **Hyperparameter tuning jobs**.

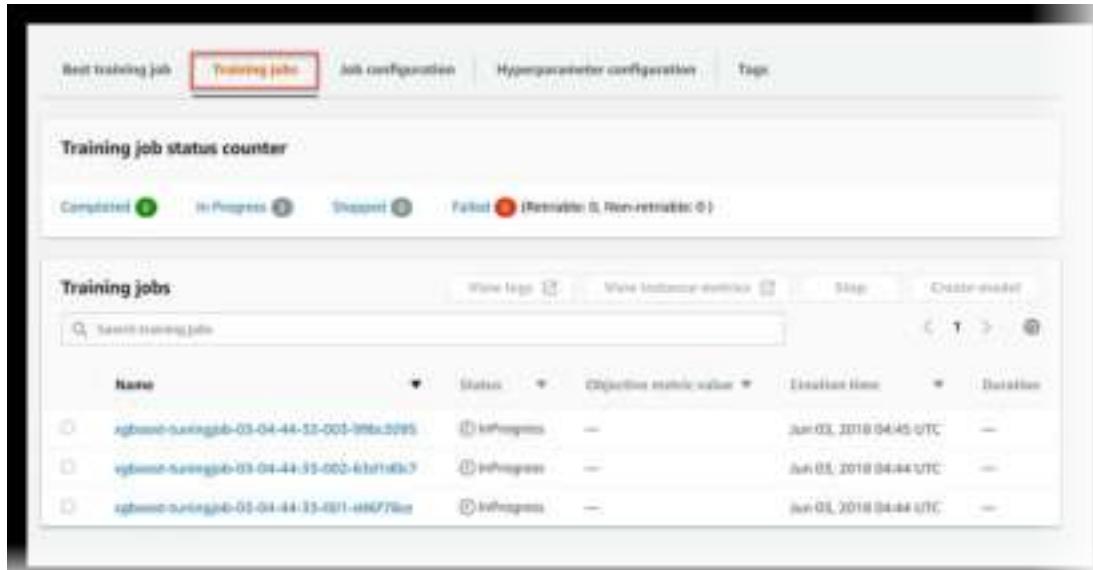


- In the list of hyperparameter tuning jobs, check the status of the hyperparameter tuning job you launched. A tuning job can be:
  - Completed**—The hyperparameter tuning job successfully completed.
  - InProgress**—The hyperparameter tuning job is in progress. One or more training jobs are still running.
  - Failed**—The hyperparameter tuning job failed.
  - Stopped**—The hyperparameter tuning job was manually stopped before it completed. All training jobs that the hyperparameter tuning job launched are stopped.
  - Stopping**—The hyperparameter tuning job is in the process of stopping.

## View the Status of the Training Jobs

### To view the status of the training jobs that the hyperparameter tuning job launched

- In the list of hyperparameter tuning jobs, choose the job that you launched.
- Choose **Training jobs**.



- View the status of each training job. To see more details about a job, choose it in the list of training jobs. To view a summary of the status of all of the training jobs that the hyperparameter tuning job launched, see **Training job status counter**.

A training job can be:

- **Completed**—The training job successfully completed.
- **InProgress**—The training job is in progress.
- **Stopped**—The training job was manually stopped before it completed.
- **Failed (Retryable)**—The training job failed, but can be retried. A failed training job can be retried only if it failed because an internal service error occurred.
- **Failed (Non-retryable)**—The training job failed and can't be retried. A failed training job can't be retried when a client error occurs.

## View the Best Training Job

A hyperparameter tuning job uses the objective metric that each training job returns to evaluate training jobs. While the hyperparameter tuning job is in progress, the best training job is the one that has returned the best objective metric so far. After the hyperparameter tuning job is complete, the best training job is the one that returned the best objective metric.

To view the best training job, choose **Best training job**.

The screenshot shows the 'Best training job' page in the Amazon SageMaker console. At the top, there are tabs: 'Best training job' (which is selected and highlighted in red), 'Training jobs', 'Job configuration', 'Hyperparameter configuration', and 'Tags'. Below the tabs, there's a section titled 'Best training job summary' with a 'Create model' button. It displays the following information:

Name	Status	Objective metric	Value
hyperparameter-tuning-job-12345-67890-1234567890	Completed	validation.auc	0.77256602048873

Below this is a section titled 'Best training job hyperparameters' with a search bar. It lists the following hyperparameters:

Name	Type	Value
_tuning_objective_metric	String	validation:auc
alpha	Continuous	1.9810243793579417
eta	Continuous	0.07404534782758304
eval_metric	String	auc

To deploy the best training job as a model that you can host at a SageMaker endpoint, choose **Create model**.

## Next Step

[Clean up \(p. 1763\)](#)

## Clean up

To avoid incurring unnecessary charges, when you are done with the example, use the Amazon Web Services Management Console to delete the resources that you created for it.

### Note

If you plan to explore other examples, you might want to keep some of these resources, such as your notebook instance, S3 bucket, and IAM role.

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/> and delete the notebook instance. Stop the instance before deleting it.
2. Open the Amazon S3 console at <https://console.amazonaws.cn/s3/> and delete the bucket that you created to store model artifacts and the training dataset.
3. Open the IAM console at <https://console.amazonaws.cn/iam/> and delete the IAM role. If you created permission policies, you can delete them, too.
4. Open the Amazon CloudWatch console at <https://console.amazonaws.cn/cloudwatch/> and delete all of the log groups that have names starting with /aws/sagemaker/.

## Stop Training Jobs Early

Stop the training jobs that a hyperparameter tuning job launches early when they are not improving significantly as measured by the objective metric. Stopping training jobs early can help reduce compute time and helps you avoid overfitting your model. To configure a hyperparameter tuning job to stop training jobs early, do one of the following:

- If you are using the Amazon SDK for Python (Boto 3), set the `TrainingJobEarlyStoppingType` field of the `HyperParameterTuningJobConfig` object that you use to configure the tuning job to `AUTO`.
- If you are using the [Amazon SageMaker Python SDK](#), set the `early_stopping_type` parameter of the `HyperParameterTuner` object to `Auto`.
- In the Amazon SageMaker console, in the **Create hyperparameter tuning job** workflow, under **Early stopping**, choose **Auto**.

For a sample notebook that demonstrates how to use early stopping, see [https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter\\_tuning/image\\_classification\\_early\\_stopping/hpo\\_image\\_classification\\_early\\_stopping.ipynb](https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/image_classification_early_stopping/hpo_image_classification_early_stopping.ipynb) or open the `hpo_image_classification_early_stopping.ipynb` notebook in the **Hyperparameter Tuning** section of the **SageMaker Examples** in a notebook instance. For information about using sample notebooks in a notebook instance, see [Example Notebooks \(p. 131\)](#).

## How Early Stopping Works

When you enable early stopping for a hyperparameter tuning job, SageMaker evaluates each training job the hyperparameter tuning job launches as follows:

- After each epoch of training, get the value of the objective metric.
- Compute the running average of the objective metric for all previous training jobs up to the same epoch, and then compute the median of all of the running averages.
- If the value of the objective metric for the current training job is worse (higher when minimizing or lower when maximizing the objective metric) than the median value of running averages of the objective metric for previous training jobs up to the same epoch, SageMaker stops the current training job.

## Algorithms That Support Early Stopping

To support early stopping, an algorithm must emit objective metrics for each epoch. The following built-in SageMaker algorithms support early stopping:

- [Linear Learner Algorithm \(p. 1442\)](#)—Supported only if you use `objective_loss` as the objective metric.
- [XGBoost Algorithm \(p. 1524\)](#)

- [Image Classification Algorithm \(p. 1399\)](#)
- [Object Detection Algorithm \(p. 1479\)](#)
- [Sequence-to-Sequence Algorithm \(p. 1512\)](#)
- [IP Insights \(p. 1410\)](#)

**Note**

This list of built-in algorithms that support early stopping is current as of December 13, 2018. Other built-in algorithms might support early stopping in the future. If an algorithm emits a metric that can be used as an objective metric for a hyperparameter tuning job (preferably a validation metric), then it supports early stopping.

To use early stopping with your own algorithm, you must write your algorithms such that it emits the value of the objective metric after each epoch. The following list shows how you can do that in different frameworks:

TensorFlow

Use the `tf.keras.callbacks.ProgbarLogger` class. For information, see [https://www.tensorflow.org/api\\_docs/python/tf/keras/callbacks/ProgbarLogger](https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ProgbarLogger).

MXNet

Use the `mxnet.callback.LogValidationMetricsCallback`. For information, see <https://mxnet.apache.org/api/python/callback/callback.html>.

Chainer

Extend chainer by using the `extensions.Evaluator` class. For information, see <https://docs.chainer.org/en/v1.24.0/reference/extensions.html#evaluator>.

PyTorch and Spark

There is no high-level support. You must explicitly write your training code so that it computes objective metrics and writes them to logs after each epoch.

## Run a Warm Start Hyperparameter Tuning Job

Use warm start to start a hyperparameter tuning job using one or more previous tuning jobs as a starting point. The results of previous tuning jobs are used to inform which combinations of hyperparameters to search over in the new tuning job. Hyperparameter tuning uses either Bayesian or random search to choose combinations of hyperparameter values from ranges that you specify. For more information, see [How Hyperparameter Tuning Works \(p. 1745\)](#). Using information from previous hyperparameter tuning jobs can help increase the performance of the new hyperparameter tuning job by making the search for the best combination of hyperparameters more efficient.

**Note**

Warm start tuning jobs typically take longer to start than standard hyperparameter tuning jobs, because the results from the parent jobs have to be loaded before the job can start. The increased time depends on the total number of training jobs launched by the parent jobs.

Reasons you might want to consider warm start include:

- You want to gradually increase the number of training jobs over several tuning jobs based on the results you see after each iteration.
- You get new data, and want to tune a model using the new data.
- You want to change the ranges of hyperparameters that you used in a previous tuning job, change static hyperparameters to tunable, or change tunable hyperparameters to static values.
- You stopped a previous hyperparameter job early or it stopped unexpectedly.

### Topics

- [Types of Warm Start Tuning Jobs \(p. 1766\)](#)
- [Warm Start Tuning Restrictions \(p. 1766\)](#)
- [Warm Start Tuning Sample Notebook \(p. 1767\)](#)
- [Create a Warm Start Tuning Job \(p. 1767\)](#)

## Types of Warm Start Tuning Jobs

There are two different types of warm start tuning jobs:

### `IDENTICAL_DATA_AND_ALGORITHM`

The new hyperparameter tuning job uses the same input data and training image as the parent tuning jobs. You can change the hyperparameter ranges to search and the maximum number of training jobs that the hyperparameter tuning job launches. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. You cannot use a new version of the training algorithm, unless the changes in the new version do not affect the algorithm itself. For example, changes that improve logging or adding support for a different data format are allowed.

Use identical data and algorithm when you use the same training data as you used in a previous hyperparameter tuning job, but you want to increase the total number of training jobs or change ranges or values of hyperparameters.

When you run an warm start tuning job of type `IDENTICAL_DATA_AND_ALGORITHM`, there is an additional field in the response to [DescribeHyperParameterTuningJob](#) named `OverallBestTrainingJob`. The value of this field is the [TrainingJobSummary](#) for the training job with the best objective metric value of all training jobs launched by this tuning job and all parent jobs specified for the warm start tuning job.

### `TRANSFER_LEARNING`

The new hyperparameter tuning job can include input data, hyperparameter ranges, maximum number of concurrent training jobs, and maximum number of training jobs that are different than those of its parent hyperparameter tuning jobs. You can also change hyperparameters from tunable to static, and from static to tunable, but the total number of static plus tunable hyperparameters must remain the same as it is in all parent jobs. The training algorithm image can also be a different version from the version used in the parent hyperparameter tuning job. When you use transfer learning, changes in the dataset or the algorithm that significantly affect the value of the objective metric might reduce the usefulness of using warm start tuning.

## Warm Start Tuning Restrictions

The following restrictions apply to all warm start tuning jobs:

- A tuning job can have a maximum of 5 parent jobs, and all parent jobs must be in a terminal state (`Completed`, `Stopped`, or `Failed`) before you start the new tuning job.
- The objective metric used in the new tuning job must be the same as the objective metric used in the parent jobs.
- The total number of static plus tunable hyperparameters must remain the same between parent jobs and the new tuning job. Because of this, if you think you might want to use a hyperparameter as tunable in a future warm start tuning job, you should add it as a static hyperparameter when you create a tuning job.
- The type of each hyperparameter (continuous, integer, categorical) must not change between parent jobs and the new tuning job.

- The number of total changes from tunable hyperparameters in the parent jobs to static hyperparameters in the new tuning job, plus the number of changes in the values of static hyperparameters cannot be more than 10. Each value in a categorical hyperparameter counts against this limit. For example, if the parent job has a tunable categorical hyperparameter with the possible values `red` and `blue`, you change that hyperparameter to static in the new tuning job, that counts as 2 changes against the allowed total of 10. If the same hyperparameter had a static value of `red` in the parent job, and you change the static value to `blue` in the new tuning job, it also counts as 2 changes.
- Warm start tuning is not recursive. For example, if you create `MyTuningJob3` as a warm start tuning job with `MyTuningJob2` as a parent job, and `MyTuningJob2` is itself a warm start tuning job with a parent job `MyTuningJob1`, the information that was learned when running `MyTuningJob1` is not used for `MyTuningJob3`. If you want to use the information from `MyTuningJob1`, you must explicitly add it as a parent for `MyTuningJob3`.
- The training jobs launched by every parent job in a warm start tuning job count against the 500 maximum training jobs for a tuning job.
- Hyperparameter tuning jobs created before October 1, 2018 cannot be used as parent jobs for warm start tuning jobs.

## Warm Start Tuning Sample Notebook

For a sample notebook that shows how to use warm start tuning, see [https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter\\_tuning/image\\_classification\\_warmstart/hpo\\_image\\_classification\\_warmstart.ipynb](https://github.com/awslabs/amazon-sagemaker-examples/blob/master/hyperparameter_tuning/image_classification_warmstart/hpo_image_classification_warmstart.ipynb). For instructions how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Example Notebooks \(p. 131\)](#). Once you have created a notebook instance and opened it, select the **SageMaker Examples** tab to see a list of all the SageMaker samples. The warm start tuning example notebook is located in the **Hyperparameter tuning** section, and is named `hpo_image_classification_warmstart.ipynb`. To open a notebook, click on its **Use** tab and select **Create copy**.

## Create a Warm Start Tuning Job

You can use either the low-level Amazon SDK for Python (Boto 3) or the high-level SageMaker Python SDK to create a warm start tuning job.

### Topics

- [Create a Warm Start Tuning Job \( Low-level SageMaker API for Python \(Boto 3\)\) \(p. 1767\)](#)
- [Create a Warm Start Tuning Job \(SageMaker Python SDK\) \(p. 1768\)](#)

### Create a Warm Start Tuning Job ( Low-level SageMaker API for Python (Boto 3))

To use warm start tuning, you specify the values of a `HyperParameterTuningJobWarmStartConfig` object, and pass that as the `WarmStartConfig` field in a call to `CreateHyperParameterTuningJob`.

The following code shows how to create a `HyperParameterTuningJobWarmStartConfig` object and pass it to `CreateHyperParameterTuningJob` job by using the low-level SageMaker API for Python (Boto 3).

Create the `HyperParameterTuningJobWarmStartConfig` object:

```
warm_start_config = {
    "ParentHyperParameterTuningJobs" : [
        {"HyperParameterTuningJobName" : 'MyParentTuningJob'}
    ],
    "WarmStartType" : "IdenticalDataAndAlgorithm"
}
```

Create the warm start tuning job:

```
smclient = boto3.Session().client('sagemaker')
smclient.create_hyper_parameter_tuning_job(HyperParameterTuningJobName =
    'MyWarmStartTuningJob',
    HyperParameterTuningJobConfig = tuning_job_config, # See notebook for tuning
    configuration
    TrainingJobDefinition = training_job_definition, # See notebook for job definition
    WarmStartConfig = warm_start_config)
```

## Create a Warm Start Tuning Job (SageMaker Python SDK)

To use the [Amazon SageMaker Python SDK](#) to run a warm start tuning job, you:

- Specify the parent jobs and the warm start type by using a `WarmStartConfig` object.
- Pass the `WarmStartConfig` object as the value of the `warm_start_config` argument of a [HyperparameterTuner](#) object.
- Call the `fit` method of the [HyperparameterTuner](#) object.

For more information about using the [Amazon SageMaker Python SDK](#) for hyperparameter tuning, see <https://github.com/aws/sagemaker-python-sdk#sagemaker-automatic-model-tuning>.

This example uses an estimator that uses the [Image Classification Algorithm \(p. 1399\)](#) algorithm for training. The following code sets the hyperparameter ranges that the warm start tuning job searches within to find the best combination of values. For information about setting hyperparameter ranges, see [Define Hyperparameter Ranges \(p. 1748\)](#).

```
hyperparameter_ranges = {'learning_rate': ContinuousParameter(0.0, 0.1),
                         'momentum': ContinuousParameter(0.0, 0.99)}
```

The following code configures the warm start tuning job by creating a `WarmStartConfig` object.

```
from sagemaker.tuner import WarmStartConfig,
                           WarmStartTypes

parent_tuning_job_name = "MyParentTuningJob"
warm_start_config = WarmStartConfig(type=WarmStartTypes.IDENTICAL_DATA_AND_ALGORITHM,
                                     parents={parent_tuning_job_name})
```

Now set the values for static hyperparameters, which are hyperparameters that keep the same value for every training job that the warm start tuning job launches. In the following code, `imageclassification` is an estimator that was created previously.

```
imageclassification.set_hyperparameters(num_layers=18,
                                         image_shape='3,224,224',
                                         num_classes=257,
                                         num_training_samples=15420,
                                         mini_batch_size=128,
                                         epochs=30,
                                         optimizer='sgd',
                                         top_k='2',
                                         precision_dtype='float32',
                                         augmentation_type='crop')
```

Now create the `HyperparameterTuner` object and pass the `WarmStartConfig` object that you previously created as the `warm_start_config` argument.

```
tuner_warm_start = HyperparameterTuner(imageclassification,
```

```
'validation:accuracy',
hyperparameter_ranges,
objective_type='Maximize',
max_jobs=10,
max_parallel_jobs=2,
base_tuning_job_name='warmstart',
warm_start_config=warm_start_config)
```

Finally, call the `fit` method of the `HyperparameterTuner` object to launch the warm start tuning job.

```
tuner_warm_start.fit(
    {'train': s3_input_train, 'validation': s3_input_validation},
    include_cls_metadata=False)
```

## Resource Limits for Automatic Model Tuning

SageMaker sets the following default limits for resources used by automatic model tuning:

- Number of parallel (concurrent) hyperparameter tuning jobs: 100
- Number of hyperparameters that can be searched: 20

**Note**

Every possible value in a categorical hyperparameter counts against this limit.

- Number of metrics defined per hyperparameter tuning job: 20
- Number of parallel (concurrent) training jobs per hyperparameter tuning job: 10

**Note**

This can be increased to one hundred.

- [Bayesian search strategy] Number of training jobs per hyperparameter tuning job: 500
- [Random search strategy] Number of training jobs per hyperparameter tuning job: 500

**Note**

This can be increased up to ten thousand.

- Maximum run time for a hyperparameter tuning job: 30 days

When you plan hyperparameter tuning jobs, you also have to take into account the limits on training resources. For information about the default resource limits for SageMaker training jobs, see [SageMaker Limits](#). Every concurrent training instance on which all of your hyperparameter tuning jobs run counts against the total number of training instances allowed. For example, if you run 10 concurrent hyperparameter tuning jobs, each of those hyperparameter tuning jobs runs 100 total training jobs and 20 concurrent training jobs. Each of those training jobs runs on one **ml.m4.xlarge** instance. The following limits apply:

- Number of concurrent hyperparameter tuning jobs: You don't need to increase the limit, because 10 tuning jobs is below the limit of 100.
- Number of training jobs per hyperparameter tuning job: You don't need to increase the limit, because 100 training jobs is below the limit of 500.
- Number of concurrent training jobs per hyperparameter tuning job: You need to request a limit increase to 20, because the default limit is 10.
- SageMaker training **ml.m4.xlarge** instances: You need to request a limit increase to 200, because you have 10 hyperparameter tuning jobs, each of which is running 20 concurrent training jobs. The default limit is 20 instances.
- SageMaker training total instance count: You need to request a limit increase to 200, because you have 10 hyperparameter tuning jobs, each of which is running 20 concurrent training jobs. The default limit is 20 instances.

**To request a quota increase:**

1. Open the [Amazon Support Center](#) page, sign in if necessary, and then choose **Create case**.
2. On the **Create case** page, choose **Service limit increase**.
3. On the **Case details** panel, select **SageMaker Automatic Model Tuning [Hyperparameter Optimization]** for the **Limit type**.
4. On the **Requests** panel for **Request 1**, select the **Region**, the resource **Limit** to increase and the **New Limit value** you are requesting. Select **Add another request** if you have additional requests for quota increases.

The screenshot shows the 'Create case' interface. In the 'Service limit increase' section, there is a note: 'Request to increase the service limit of your resources'. Below this, under 'Case details', the 'Limit type' is set to 'SageMaker Automatic Model Tuning [Hyperparameter Optimization]'. In the 'Requests' section, there is a note: '(?) To request additional limit increases for the same limit type, choose Add another request. To request an increase for a different limit type, create a separate limit increase request.' A single request is listed: 'Request 1' with 'Region' set to 'US West (Oregon)', 'Resource Type' set to 'SageMaker Automatic Model Tuning [Hyperparameter Optimization]', and 'Limit' set to 'Max Parallel Training Jobs Per Tuning Job'. The 'New Limit value' field contains '100'. At the bottom of the Requests section is a button labeled 'Add Another Request'.

5. In the **Case description** panel, provide a description of your use case.
6. In the **Contact options** panel, select your preferred **Contact methods (Web, Chat or Phone)** and then choose **Submit**.

## Best Practices for Hyperparameter Tuning

Hyperparameter optimization is not a fully-automated process. To improve optimization, use the following guidelines when you create hyperparameters.

### Topics

- [Choosing the Number of Hyperparameters \(p. 1771\)](#)
- [Choosing Hyperparameter Ranges \(p. 1771\)](#)
- [Using Logarithmic Scales for Hyperparameters \(p. 1771\)](#)
- [Choosing the Best Number of Concurrent Training Jobs \(p. 1771\)](#)

- [Running Training Jobs on Multiple Instances \(p. 1771\)](#)

## Choosing the Number of Hyperparameters

The computational complexity of a hyperparameter tuning job depends primarily on the number of hyperparameters whose range of values Amazon SageMaker has to search through during optimization. Although you can simultaneously specify up to 20 hyperparameters to optimize for a tuning job, limiting your search to a much smaller number is likely to give you better results.

## Choosing Hyperparameter Ranges

The range of values for hyperparameters that you choose to search can significantly affect the success of hyperparameter optimization. Although you might want to specify a very large range that covers every possible value for a hyperparameter, you get better results by limiting your search to a small range of values. If you know that you get the best metric values within a subset of the possible range, consider limiting the range to that subset.

## Using Logarithmic Scales for Hyperparameters

During hyperparameter tuning, SageMaker attempts to figure out if your hyperparameters are log-scaled or linear-scaled. Initially, it assumes that hyperparameters are linear-scaled. If they are in fact log-scaled, it might take some time for SageMaker to discover that fact. If you know that a hyperparameter is log-scaled and can convert it yourself, doing so could improve hyperparameter optimization.

## Choosing the Best Number of Concurrent Training Jobs

When setting the resource limit `MaxParallelTrainingJobs` for the maximum number of concurrent training jobs that a hyperparameter tuning job can launch, consider the following tradeoff. Running more hyperparameter tuning jobs concurrently gets more work done quickly, but a tuning job improves only through successive rounds of experiments. Typically, running one training job at a time achieves the best results with the least amount of compute time.

## Running Training Jobs on Multiple Instances

When a training job runs on multiple instances, hyperparameter tuning uses the last-reported objective metric value from all instances of that training job as the value of the objective metric for that training job. Design distributed training jobs so that the objective metric reported is the one that you want.

# Distributed Training

## Topics

- [Get Started with Distributed Training \(p. 1772\)](#)
- [Basic Distributed Training Concepts \(p. 1772\)](#)
- [Advanced Concepts \(p. 1773\)](#)
- [Strategies \(p. 1774\)](#)
- [Optimize Distributed Training \(p. 1775\)](#)
- [Scenarios \(p. 1776\)](#)
- [SageMaker Built-In Distributed Training Features \(p. 1779\)](#)
- [SageMaker's Distributed Data Parallel Library \(p. 1779\)](#)
- [SageMaker's Distributed Model Parallel \(p. 1795\)](#)

- [Distributed Training Jupyter Notebook Examples \(p. 1819\)](#)

## Get Started with Distributed Training

If you're familiar with distributed training, follow one of the links to your preferred strategy or framework to get started. Otherwise, continue on to the next section to learn some distributed training concepts.

### SageMaker distributed training libraries:

- [SageMaker's Distributed Data Parallel Library \(p. 1779\)](#)
- [SageMaker's Distributed Model Parallel \(p. 1795\)](#)

## Basic Distributed Training Concepts

SageMaker's distributed training libraries use the following distributed training terms and features.

### Datasets and Batches

- **Training Dataset:** All of the data you use to train the model.
- **Global batch size:** The number of records selected from the training dataset in each iteration to send to the GPUs in the cluster. This is the number of records over which the gradient is computed at each iteration. If data parallelism is used, it is equal to the total number of model replicas multiplied by the per-replica batch size: global batch size = # model replicas x per-replica batch size. A single batch of global batch size is often referred to as the *mini-batch* in machine learning literature.
- **Per-replica batch size:** When data parallelism is used, this is the number of records sent to each model replica. Each model replica performs a forward and backward pass with this batch to calculate weight updates. The resulting weight updates are synchronized (averaged) across all replicas before the next set of per-replica batches are processed.
- **Micro-batch:** A subset of the mini-batch or, if hybrid model and data parallelism is used , it is a subset of the per-replica sized batch . When you use SageMaker's distributed model parallelism library, each micro-batch is fed into the training pipeline one-by-one and follows an [execution schedule](#) defined by the library's runtime.

### Training

- **Epoch:** One training cycle through the entire dataset. It is common to have multiple iterations per an epoch. The number of epochs you use in training is unique on your model and use case.
- **Iteration:** A single forward and backward pass performed using a global batch sized batch (a mini-batch) of training data. The number of iterations performed during training is determined by the global batch size and the number of epochs used for training. For example, if a dataset includes 5,000 samples, and you use a global batch size of 500, it will take 10 iterations to complete a single epoch.
- **Learning rate:** A variable that influences the amount that weights are changed in response to the calculated error of the model. The learning rate plays an important role in the model's ability to converge as well as the speed and optimality of convergence.

### Instances and GPUs

- **Instances:** An Amazon [machine learning compute instance](#). These are also referred to as *nodes*.
- **Cluster size:** When using SageMaker's distributed training library, this is the number of instances multiplied by the number of GPUs in each instance. For example, if you use two ml.p3.8xlarge instances in a training job, which have 4 GPUs each, the cluster size is 8. While increasing cluster size can lead to faster training times, communication between instances must be optimized; Otherwise,

communication between the nodes can add overhead and lead to slower training times. The SageMaker distributed training library is designed to optimize communication between Amazon ML compute instances, leading to higher device utilization and faster training times.

### Distributed Training Solutions

- **Data parallelism:** A strategy in distributed training where a training dataset is split up across multiple processing nodes (such as Amazon ML Instances), and each processing node contains a *replica* of the model. Each node receives different batches of training data, performs a forward and backward pass, and shares weight updates with the other nodes for synchronization before moving on to the next batch and ultimately another epoch.
- **Model parallelism:** A strategy in distributed training where the model partitioned across multiple processing nodes (such as Amazon ML Instances). The model might be complex and have a large number of hidden layers and weights, making it unable to fit in the memory of a single node. Each node carries a subset of the model, through which the data flows and the transformations are shared and compiled. The efficiency of model parallelism, in terms of GPU utilization and training time, is heavily dependent on how the model is partitioned and the execution schedule used to perform forward and backward passes.
- **Pipeline Execution Schedule (Pipelining):** The pipeline execution schedule determines the order in which computations (micro-batches) are made and data is processed across devices during model training. Pipelining is a technique to achieve true parallelization in model parallelism and overcome the performance loss due to sequential computation by having the GPUs compute simultaneously on different data samples. To learn more, see [Pipeline Execution Schedule](#).

### Example

The following example demonstrates how these terms might be used to describe a training job that uses hybrid model and data parallelism:

A 2-way data parallelism and 4-way model parallelism training job is launched with a global batch size of 64 images. This training job requires a total of 8 GPUs. Each of the two model replicas processes a per-replica batch of size 32. During the forward and backward pass, a per-replica batch is further divided up into micro-batches. These micro-batches are processed in an interleaved fashion using a pipelined execution schedule.

The training dataset includes 640 images, and so a single epoch takes 10 iterations.

## Advanced Concepts

Machine Learning (ML) practitioners commonly face two scaling challenges when training models: *scaling model size* and *scaling training data*. While model size and complexity can result in better accuracy, there is a limit to the model size you can fit into a single CPU or GPU. Furthermore, scaling model size may result in more computations and longer training times.

Not all models handle training data scaling equally well. Some algorithms need to ingest all the training data *in memory* for training. They only scale vertically, and to bigger and bigger instance types. Some algorithms have the disadvantage of scaling super-linearly with dataset size. For example, the computational load of a nearest-neighbor search for all records of an N-rows dataset scales in N squared. In most cases, scaling training data results in longer training times.

Deep Learning (DL) is a specific family of ML algorithms consisting of several layers of artificial neural networks. The most common training method is with mini-batch Stochastic Gradient Descent (SGD). In mini-batch SGD, the model is trained by conducting small iterative changes of its coefficients in the direction that reduces its error. Those iterations are conducted on equally sized subsamples of the training dataset called *mini-batches*. For each mini-batch, the model is run in each record of the mini-batch, its error measured and the gradient of the error estimated. Then the average gradient is measured

across all the records of the mini-batch and provides an update direction for each model coefficient. One full pass over the training dataset is called an *epoch*. Model trainings commonly consist of dozens to hundreds of epochs. Mini-batch SGD has several benefits: First, its iterative design makes training time theoretically linear of dataset size. Second, in a given mini-batch each record is processed individually by the model without need for inter-record communication other than the final gradient average. The processing of a mini-batch is consequently particularly suitable for parallelization and distribution.

Parallelizing SGD training by distributing the records of a mini-batch over different computing devices is called *data parallel distributed training*, and is the most commonly used DL distribution paradigm. Data parallel training is a relevant distribution strategy to scale the mini-batch size and process each mini-batch faster. However, data parallel training comes with the extra complexity of having to compute the mini-batch gradient average with gradients coming from all the workers and communicating it to all the workers, a step called *allreduce* that can represent a growing overhead, as the training cluster is scaled, and that can also drastically penalize training time if improperly implemented or implemented over improper hardware substracts.

Data parallel SGD still requires developers to be able to fit at least the model and a single record in a computing devices, like a single CPU or GPU. When training very large models such as large transformers in Natural Language Processing (NLP), or segmentation models over high-resolution images, there may be situations in which this is not feasible. An alternative way to break up the workload is to partition the model over multiple computing devices, an approach called *model-parallel distributed training*.

## Strategies

Distributed training is usually split by two approaches: data parallel and model parallel. *Data parallel* is the most common approach to distributed training: You have a lot of data, batch it up, and send blocks of data to multiple CPUs or GPUs (nodes) to be processed by the neural network or ML algorithm, then combine the results. The neural network is the same on each node. A *model parallel* approach is used with large models that won't fit in a node's memory in one piece; it breaks up the model and places different parts on different nodes. In this situation, you need to send your batches of data out to each node so that the data is processed on all parts of the model.

The terms *network* and *model* are often used interchangeably: A large model is really a large network with many layers and parameters. Training with a large network produces a large model, and loading the model back onto the network with all your pre-trained parameters and their weights loads a large model into memory. When you break apart a model to split it across nodes, you're also breaking apart the underlying network. A network consists of layers, and to split up the network, you put layers on different compute devices.

A common pitfall of naively splitting layers across devices is severe GPU under-utilization. Training is inherently sequential in both forward and backward passes, and at a given time, only one GPU can actively compute, while the others wait on the activations to be sent. Modern model parallel libraries solve this problem by using pipeline execution schedules to improve device utilization. However, only the Amazon SageMaker's distributed model parallel library includes automatic model splitting. The two core features of the library, automatic model splitting and pipeline execution scheduling, simplifies the process of implementing model parallelism by making automated decisions that lead to efficient device utilization.

## Train with Data Parallel and Model Parallel

If you are training with a large dataset, start with a data parallel approach. If you run out of memory during training, you may want to switch to a model parallel approach, or try hybrid model and data parallelism. You can also try the following to improve performance with data parallel:

- Change your model's hyperparameters.
- Reduce the batch size.
- Keep reducing the batch size until it fits. If you reduce batch size to 1, and still run out of memory, then you should try model-parallel training.

Try gradient compression (fp16, fp8):

- On NVIDIA TensorCore-equipped hardware, using [mixed-precision training](#) creates both speed-up and memory consumption reduction.

Try reducing the input size:

- Reduce the NLP sequence length if you increase the sequence link, need to adjust the batch size down, or adjust the GPUs up to spread the batch.
- Reduce image resolution.

Check if you use batch normalization, since this can impact convergence. When you use distributed training, your batch is split across GPUs and the effect of a much lower batch size can be a higher error rate thereby disrupting the model from converging. For example, if you prototyped your network on a single GPU with a batch size of 64, then scaled up to using four p3dn.24xlarge, you now have 32 GPUs and your per-GPU batch size drops from 64 to 2. This will likely break the convergence you saw with a single node.

Start with model-parallel training when:

- Your model does not fit on a single device.
- Due to your model size, you're facing limitations in choosing larger batch sizes, such as if your model weights take up most of your GPU memory and you are forced to choose a smaller, suboptimal batch size.

To learn more about the SageMaker distributed libraries, see the following:

- [SageMaker's Distributed Data Parallel Library \(p. 1779\)](#)
- [SageMaker's Distributed Model Parallel \(p. 1795\)](#)

## Optimize Distributed Training

Customize hyperparameters for your use case and your data to get the best scaling efficiency. In the following discussion, we highlight some of the most impactful training variables and provide references to state-of-the-art implementations so you can learn more about your options. Also, we recommend that you refer to your preferred framework's distributed training documentation.

- [Apache MXNet distributed training](#)
- [PyTorch distributed training](#)
- [TensorFlow distributed training](#)

## Batch Size

SageMaker distributed toolkits generally allow you to train on bigger batches. For example, if a model fits within a single device but can only be trained with a small batch size, using either model-parallel training or data parallel training enables you to experiment with larger batch sizes.

Be aware that batch size directly influences model accuracy by controlling the amount of noise in the model update at each iteration. Increasing batch size reduces the amount of noise in the gradient estimation, which can be beneficial when increasing from very small batches sizes, but can result in degraded model accuracy as the batch size increases to large values.

### Tip

Adjust your hyperparameters to ensure that your model trains to a satisfying convergence as you increase its batch size.

A number of techniques have been developed to maintain good model convergence when batch is increased.

## Mini-Batch Size

In SGD, the mini-batch size quantifies the amount of noise present in the gradient estimation. A small mini-batch results in a very noisy mini-batch gradient, which is not representative of the true gradient over the dataset. A large mini-batch results in a mini-batch gradient close to the true gradient over the dataset and potentially not noisy enough—likely to stay locked in irrelevant minima.

To learn more about these techniques, see the following papers:

- [Accurate, Large Minibatch SGD:Training ImageNet in 1 Hour](#), Goya et al.
- [PowerAI DDL](#), Cho et al.
- [Scale Out for Large Minibatch SGD: Residual Network Training on ImageNet-1K with Improved Accuracy and Reduced Time to Train](#), Codreanu et al.
- [ImageNet Training in Minutes](#), You et al.
- [Large Batch Training of Convolutional Networks](#), You et al.
- [Large Batch Optimization for Deep Learning: Training BERT in 76 Minutes](#), You et al.
- [Accelerated Large Batch Optimization of BERT Pretraining in 54 minutes](#), Zheng et al.
- [Deep Gradient Compression](#), Lin et al.

## Scenarios

The following sections cover scenarios in which you may want to scale up training, and how you can do so using Amazon resources.

### Scaling from a Single GPU to Many GPUs

The amount of data or the size of the model used in machine learning can create situations in which the time to train a model is longer than you are willing to wait. Sometimes, the training doesn't work at all because the model or the training data is too large. One solution is to increase the number of GPUs you use for training. On an instance with multiple GPUs, like a p3.16xlarge that has eight GPUs, the data and processing is split across the eight GPUs. When you use distributed training libraries, this can result in a near-linear speedup in the time it takes to train your model. It takes slightly over 1/8 the time it would have taken on p3.2xlarge with one GPU.

Instance type	GPUs
p3.2xlarge	1
p3.8xlarge	4
p3.16xlarge	8
p3dn.24xlarge	8

#### Note

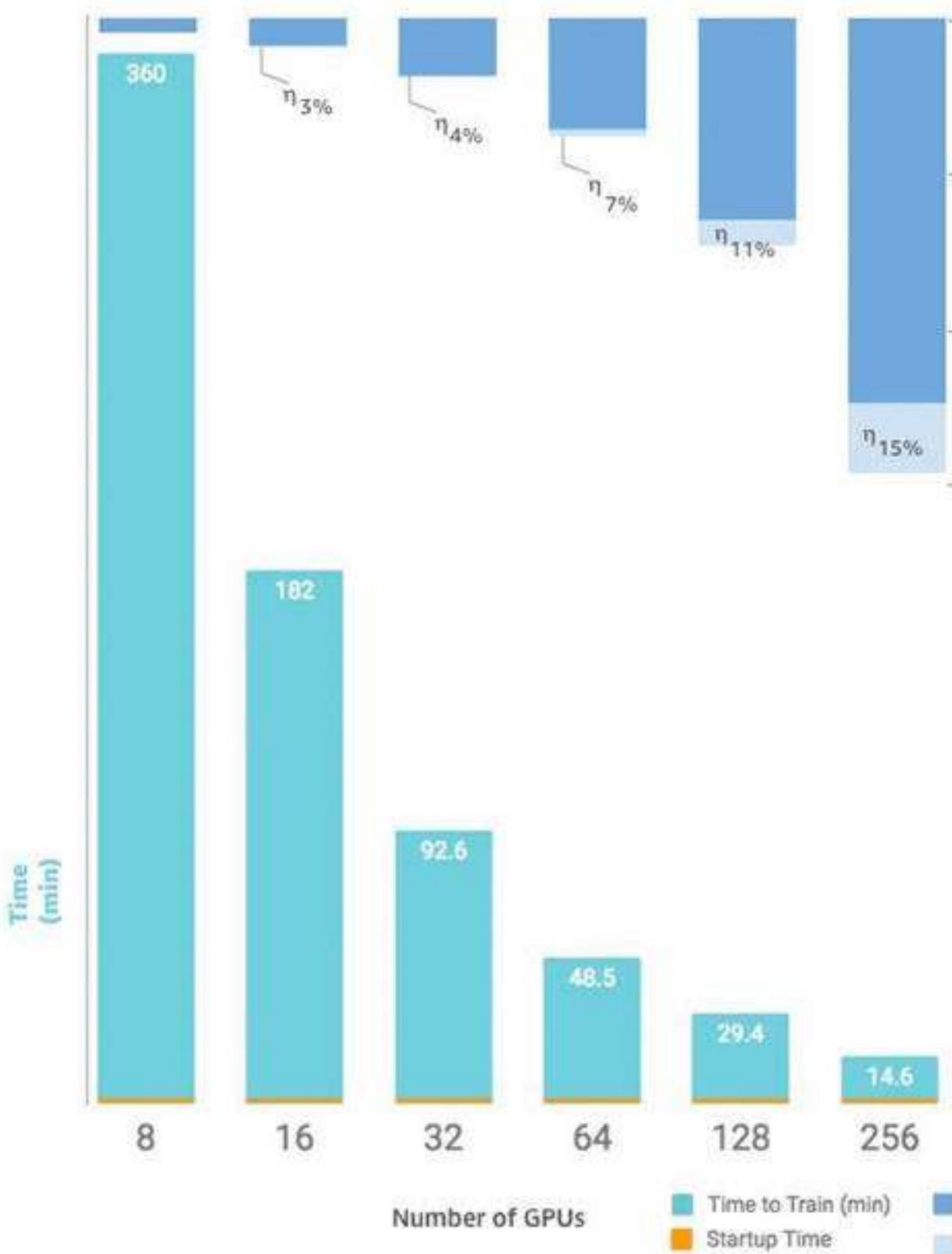
The ml instance types used by SageMaker training have the same number of GPUs as the corresponding p3 instance types. For example, ml.p3.8xlarge has the same number of GPUs as p3.8xlarge - 4.

## Scaling from a Single Instance to Multiple Instances

If you want to scale your training even further, you can use more instances. However, you should choose a larger instance type before you add more instances. Review the previous table to see how many GPUs are in each p3 instance type.

If you have made the jump from a single GPU on a p3.2xlarge to four GPUs on a p3.8xlarge, but decide that you require more processing power, you may see better performance and incur lower costs if you choose a p3.16xlarge before trying to increase instance count. Depending on the libraries you use, when you keep your training on a single instance, performance is better and costs are lower than a scenario where you use multiple instances.

When you are ready to scale the number of instances, you can do this with SageMaker Python SDK's `estimator` function by setting your `instance_count`. For example, you can set `instance_type = p3.16xlarge` and `instance_count = 2`. Instead of the eight GPUs on a single p3.16xlarge, you have 16 GPUs across two identical instances. The following chart shows [scaling and throughput starting with eight GPUs](#) on a single instance and increasing to 64 instances for a total of 256 GPUs.



## Availability Zones and Network Backplane

With multiple instances, it's important to understand the network that connects the instances, how they read the training data, and how they share information between themselves (for example, communication between the nodes in the cluster when doing an AllReduce operation).

First, your instances need to be in the same Region and same Availability Zone. For example, instances in us-west-2 must all be in us-west-2a. When you use the SageMaker Python SDK, this is handled for you. If you use Amazon EC2 and orchestrate your own training clusters, you need to be aware of this, or your training speeds suffer.

Your training data should also be in the same Availability Zone. When you use a SageMaker estimator, you pass in the Region and the S3 bucket, and if the data is not in the Region you set, you get an error.

## Optimized GPU, Network, and Storage

The p3dn.24xlarge instance type was designed for fast local storage and a fast network backplane with up to 100 gigabits, and we highly recommend it as the most performant option for distributed training. SageMaker supports streaming data modes from S3, referred to as pipe mode. For HPC loads like distributed training, we recommend [Amazon Fsx](#) for your file storage.

## Custom Training Scripts

While SageMaker makes it simple to deploy and scale the number of instances and GPUs, depending on your framework of choice, managing the data and results can be very challenging, which is why external supporting libraries are often used. This most basic form of distributed training requires modification of your training script to manage the data distribution.

SageMaker also supports Horovod and implementations of distributed training native to each major deep learning framework. If you choose to use examples from these frameworks, you can follow SageMaker's [container guide](#) for Deep Learning Containers, and various [example notebooks](#) that demonstrate implementations.

## SageMaker Built-In Distributed Training Features

The [SageMaker built-in libraries of algorithms](#) consists of 18 popular machine learning algorithms. Many of them were rewritten from scratch to be scalable and distributed out of the box. If you want to use distributed **deep learning** training code, we recommend Amazon SageMaker's distributed training libraries. SageMaker's distributed training libraries make it easier for you to write highly scalable and cost-effective custom data parallel and model parallel deep learning training jobs.

SageMaker distributed training libraries offer both data parallel and model parallel training strategies. It combines software and hardware technologies to improve inter-GPU and inter-node communications. It extends SageMaker's training capabilities with built-in options that require only small code changes to your training scripts.

- [SageMaker's Distributed Data Parallel Library \(p. 1779\)](#)
- [SageMaker's Distributed Model Parallel \(p. 1795\)](#)

## SageMaker's Distributed Data Parallel Library

### Important

To use new features with an existing notebook instance or Studio application, restart the notebook instance or Studio application to get the latest updates.

SageMaker's distributed data parallel library extends SageMaker's training capabilities on deep learning models with near-linear scaling efficiency, achieving fast time-to-train with minimal code changes.

- The library optimizes your training job for Amazon network infrastructure and Amazon EC2 instance topology.
- The library takes advantage of gradient updates to communicate between nodes with a custom AllReduce algorithm.

When training a model on a large amount of data, machine learning practitioners often turn to distributed training to reduce the time to train. In some cases, where time is of the essence, the business requirement is to finish training as quickly as possible or at least within a constrained time period. Then, distributed training is scaled to use a cluster of multiple nodes—not just multiple GPUs in a computing instance, but multiple instances with multiple GPUs. As the cluster size increases, so does the significant drop in performance. This drop in performance is primarily caused by the communications overhead between nodes in a cluster.

SageMaker's distributed library offers two options for distributed training: model parallel and data parallel. This guide focuses on how to train models using a data parallel strategy. For more information on training with a model-parallel strategy, refer to [SageMaker's Distributed Model Parallel \(p. 1795\)](#).

### Topics

- [Introduction to SageMaker's Distributed Data Parallel Library \(p. 1780\)](#)
- [Use the SageMaker Distributed Data Parallel API \(p. 1783\)](#)
- [Modify Your Training Script using the SageMaker Data Parallel Library \(p. 1787\)](#)
- [SageMaker distributed data parallel Configuration Tips and Pitfalls \(p. 1792\)](#)
- [Data Parallel Library FAQ \(p. 1793\)](#)
- [Data Parallel Troubleshooting \(p. 1794\)](#)

## Introduction to SageMaker's Distributed Data Parallel Library

### Why Use SageMaker Distributed Data Parallel Library?

SageMaker's distributed data parallel library addresses communications overhead in two ways:

1. The library performs AllReduce, a key operation during distributed training that is responsible for a large portion of communication overhead.
2. The library performs optimized node-to-node communication by fully utilizing Amazon's network infrastructure and Amazon EC2 instance topology.

Use this data parallel library to increase speed by up to 25% in training models such as BERT. While implementations like Horovod offer sub-linear performance at scale, this library offers near-linear performance at scale. This means that you get a faster training time and a lower cost to train a model.

### Training Benchmarks

#### PyTorch with SageMaker's data parallel library

Model	Setup	Throughput			Scaling Efficiency		
		PT-DDP	SageMaker	Speed up	PT-DDP	SageMaker	Improvement
BERT Large (finetuned)	2 node p3dn.24xlarge	1752	2479	41%	66%	90%	20%
	4 node p3dn.24xlarge	3017	4903	52%	55%	84%	38%
	8 node p3dn.24xlarge	7429	8581	13%	67%	78%	11%
MaskRCNN (finetuned)	2 node p3dn.24xlarge	152	158	4%	10%	85%	3%
	4 node p3dn.24xlarge	258	307	19%	10%	83%	15%
	8 node p3dn.24xlarge	545	617	13%	74%	84%	9%

Using instance type p3dn.24xlarge and on 2, 4, and 8 node clusters:

- *BERT*: When used with PyTorch, the SageMaker library is 41%, 52%, and 13% faster than PyTorch-DDP.
- *MaskRCNN*: When used with PyTorch, the SageMaker library is 4%, 19%, and 15% faster than PyTorch-DDP.

These benchmarks were run on PyTorch v1.6 using m1.p3dn.24xlarge instances. You can find the training code on the [SageMaker examples website](#). The examples website also has benchmark training code for these models [using TensorFlow 2.3](#).

## Optimal Bandwidth Use with Balanced Fusion Buffer

SageMaker's distributed data parallel library uses a communication pattern similar to parameter servers to reduce the amount of data transferred and the number of steps involved in averaging gradients from multiple GPUs. It also uses a new technique called balanced fusion buffers to make optimal use of the bandwidth available across all nodes in the cluster.

One key disadvantage of traditional parameter servers is their suboptimal use of available network bandwidth. Parameter servers treat variables as atomic units and place each variable on one server. Since gradients become available sequentially during the backward pass, at any given instant, there is imbalance in the volume of data being sent and received from different servers. Some servers are receiving and sending more data, some less, and some none. This problem becomes worse as the number of parameter servers increases.

The library addresses these problems by introducing *balanced fusion buffers*. A balanced fusion buffer is a buffer in the GPU that holds the gradients until the size of the buffer exceeds a threshold. In a setup with N parameter servers, when the buffer exceeds the threshold, the balanced fusion buffer is copied to CPU memory, sharded into N parts, and the ith part is sent to the ith parameter server. Each server receives exactly the same number of bytes from a balanced fusion buffer. The ith server receives the ith partition of the balanced fusion buffer from all workers, sums them up, and sends the results back to all workers. Since all the servers participate equally in averaging each balanced fusion buffer, server bandwidth is efficiently utilized.

## Optimal GPU Usage with Efficient AllReduce Overlapping with a Backward Pass

SageMaker's distributed data parallel library achieves optimal overlapping of the AllReduce operation with the backward pass, significantly improving the GPU utilization, and achieves near-linear scaling efficiency and faster time to train by optimizing tasks between CPUs and GPUs. The library performs AllReduce in parallel while GPU is computing gradients without taking away additional GPU cycles, which makes the library faster.

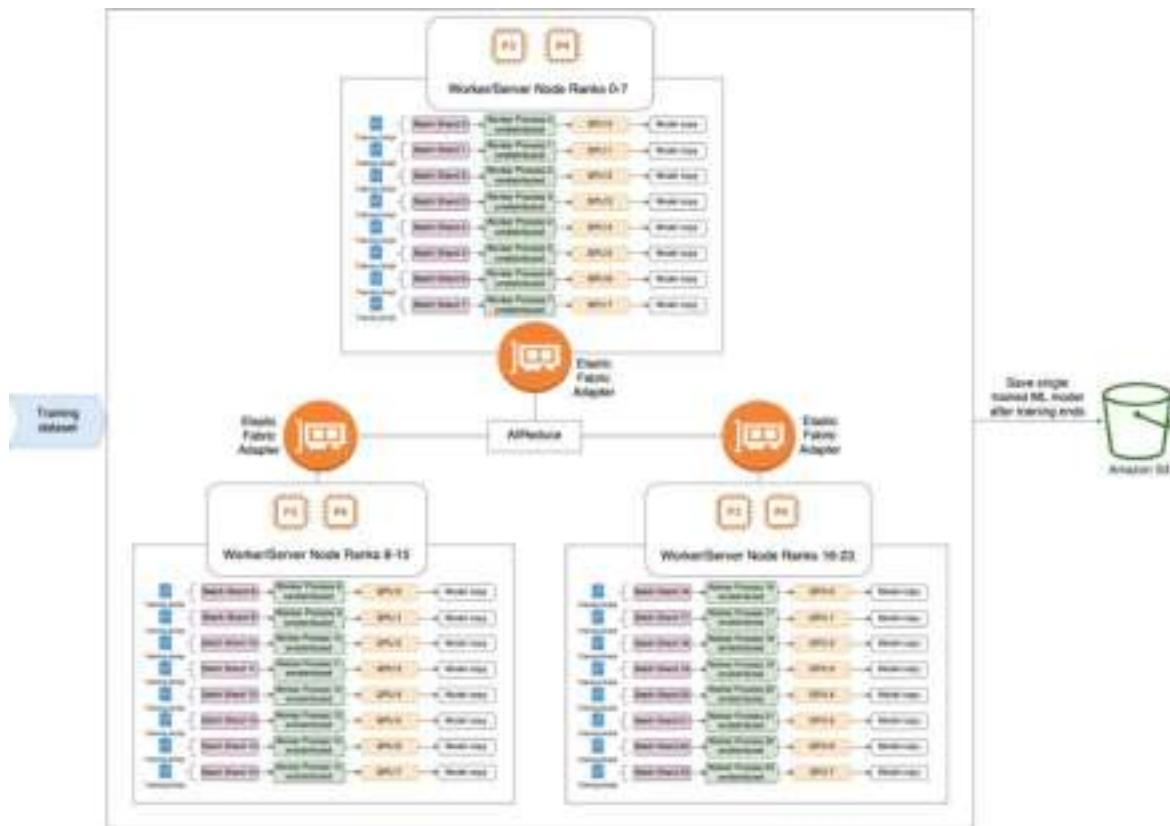
- *Leverages CPUs:* The library uses CPUs to AllReduce gradients, offloading this task from the GPUs.
- *Improved GPU usage:* The cluster's GPUs focus on computing gradients, improving their utilization throughout training.

## SageMaker Distributed Data Parallel Architecture

The library supports larger compute instances that have 8 GPUs per node: `m1.p3.16xlarge`, `m1.p3dn.24xlarge`, and `m1.p4d.24xlarge`. The high-level workflow of the SageMaker distributed data parallel library is as following:

1. The library assigns ranks to GPUs (workers).
2. At each iteration, the library divides each global batch by the total number of workers (world size) and assigns small batches (batch shards) to the workers.
  - The size of the global batch is `(number of nodes in a cluster) * (number of GPUs per node) * (per batch shard)`.
  - A batch shard (small batch) is a subset of dataset assigned to each GPU (worker) per iteration.
3. The library launches a training script on each worker.
4. The library manages copies of model weights and gradients from the workers at the end of every iteration.
5. The library synchronizes model weights and gradients across the workers to aggregate a single trained model.

The following architecture diagram shows an example of how the library sets up data parallelism for a cluster of 3 nodes.



To start using the SageMaker distributed data parallel library, see [Use the SageMaker Distributed Data Parallel API \(p. 1783\)](#) to set up a SageMaker estimator through [Amazon SageMaker Python SDK](#), and [Modify Your Training Script using the SageMaker Data Parallel Library \(p. 1787\)](#) to adapt your training script using the SageMaker distributed data parallel library.

## Use the SageMaker Distributed Data Parallel API

Use this page to learn how you can import and use SageMaker's distributed data parallel library. For API specifications, refer to [distributed data parallel](#) in the SageMaker Python SDK documentation.

### Use SageMaker's Distributed Data Parallel Library

To use SageMaker's distributed data parallel library, you start with a training script. Just as with any training job on SageMaker, you use your training script to launch a SageMaker training job. To get started, we recommend that you use the SageMaker data parallel library in the following ways:

- Using a [SageMaker notebook instance](#)
- Using [SageMaker Studio](#)

With either option, you can then use the [Distributed Training Jupyter Notebook Examples \(p. 1819\)](#) provided with this documentation to get started.

### Use the Data Parallel Library with SageMaker's Python SDK

To use SageMaker's distributed data parallel library, you must create a training script for one of the supported frameworks and launch the training job using the SageMaker Python SDK. To learn how you can incorporate the library into a training script, see [Modify Your Training Script Using SageMaker's Distributed Model Parallel Library \(p. 1804\)](#). The library API documentation is located in the SageMaker Python SDK. See SageMaker's [data parallel API documentation](#).

SageMaker supports the following training environment configurations:

- You can use a prebuilt TensorFlow or PyTorch container.
- You can customize SageMaker prebuilt containers or extend them to handle any additional functional requirements for your algorithm or model that the prebuilt SageMaker Docker image doesn't support. For an example of how you can extend a pre-built container, see [Extend a Prebuilt Container](#).

To extend a pre-built container, or adapt your own container to use the library, you must use one of the PyTorch or TensorFlow GPU general framework base-images. The distributed data parallel library is included in all CUDA 11 (cu11x) TensorFlow 2.3.x and PyTorch 1.6.x versioned images and later. See [Available Deep Learning Containers Images](#) for a list of available images.

#### Important

It is recommended that you use the image that contains the latest version of TensorFlow or PyTorch to access the most up to date version of the SageMaker distributed data parallel library. All images that include TensorFlow 2.4.1 and PyTorch 1.8.1 versions and later support EFA instance types (`ml.p3dn.24xlarge`, `ml.p4d.24xlarge`).

For example, if you are using PyTorch 1.8.1, your Dockerfile should contain a `FROM` statement similar to the following:

```
# SageMaker PyTorch image
FROM 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-training:1.8.1-gpu-py36-cu111-
ubuntu18.04

ENV PATH="/opt/ml/code:${PATH}"
```

```
# this environment variable is used by the SageMaker PyTorch container to determine our
# user code directory.
ENV SAGEMAKER_SUBMIT_DIRECTORY /opt/ml/code

# /opt/ml and all subdirectories are utilized by SageMaker, use the /code subdirectory to
# store your user code.
COPY cifar10.py /opt/ml/code/cifar10.py

# Defines cifar10.py as script entrypoint
ENV SAGEMAKER_PROGRAM cifar10.py
```

- You can adapt your own Docker container to work with SageMaker using the [SageMaker Training toolkit](#). For an example, see [Adapting Your Own Training Container](#).

In all cases, you launch your job using a SageMaker Python SDK Estimator.

If you are using the SageMaker pre-built PyTorch or TensorFlow containers, see the following section to learn how configure an Estimator to use the library.

If you bring your own container or extend a pre-built container, you can create an instance of the Estimator class with the desired Docker image.

## TensorFlow Estimator

You can use SageMaker's [distributed data parallel library API](#) by specifying `smdistributed.dataparallel` as the distribution strategy in the SageMaker TensorFlow Estimator API. Refer to [SageMaker TensorFlow Estimator API documentation](#) for full details about the API.

```
class sagemaker.tensorflow.estimator.TensorFlow(py_version=None, framework_version=None, model_dir=None)
```

The following two parameters of the SageMaker Python SDK TensorFlow estimator are required to use `smdistributed.dataparallel` with TensorFlow.

**distribution(dict)(optional)**: A dictionary with information on how to run distributed training (default: None).

- To use `smdistributed.dataparallel` as a distribution strategy, use the following option:

```
distribution = { "smdistributed": { "dataparallel": { "enabled": True } } }
```

- **custom\_mpi\_options (str)(optional)**: Custom MPI options. The following is an example of how you can use this parameter when defining distribution. To learn more, see [Custom MPI Options \(p. 1793\)](#).

```
distribution = {
    "smdistributed": {
        "dataparallel": {
            "enabled": True,
            "custom_mpi_options": "-verbose -x NCCL_DEBUG=VERSION"
        }
    }
}
```

**train\_instance\_type (str)(required)**: A type of Amazon EC2 instance to use.

- If the `smdistributed.dataparallel` distribution strategy is used, the allowed instance types are: `ml.p4d.24xlarge`, `ml.p3dn.24xlarge`, and `ml.p3.16xlarge`. For best performance, it is

recommended that you use an EFA supported instance, `ml.p3dn.24xlarge` or `ml.p4d.24xlarge`, with the latest supported version of TensorFlow.

### Example

```
from sagemaker.tensorflow import TensorFlow

tf_estimator = TensorFlow(
    base_job_name = "training_job_name_prefix",
    entry_point="tf-train.py",
    role="SageMakerRole",
    framework_version="2.4.1",
    # You must set py_version to py36
    py_version="py37",
    # For training with multi node distributed training, set this count.
    # Example: 2,3,4,..8
    instance_count=2,
    # For training with p3dn instance use - ml.p3dn.24xlarge
    instance_type="ml.p3.16xlarge",
    # Training using smdistributed.dataparallel Distributed Training Framework
    distribution={"smdistributed": {"dataparallel": {"enabled": True}}}
)

tf_estimator.fit("s3://bucket/path/to/training/data")
```

### Note

If you are using the SageMaker Python SDK 1.x, you need to use hyperparameters to specify to use `smdistributed.dataparallel` as the distributed training strategy.

```
from sagemaker.tensorflow import TensorFlow
tf_estimator = TensorFlow(
    ...
    # Training using smdistributed.dataparallel Distributed
    Training Framework

    hyperparameters={"sagemaker_distributed_dataparallel_enabled": True},
    ...
)
tf_estimator.fit("s3://bucket/path/to/training/data")
```

## PyTorch Estimator

You can use SageMaker's [distributed data parallel library API](#) by specifying `smdistributed.dataparallel` as the distribution strategy in SageMaker PyTorch Estimator API. Refer to the [SageMaker PyTorch Estimator API specification](#) for full details of the API.

```
class sagemaker.pytorch.estimator.PyTorch(py_version=None, framework_version=None, model_dir=None, image_name=None, hyperparameters=None, distribution=None, kwargs)
```

The following two parameters of the SageMaker PyTorch Estimator are necessary for using `smdistributed.dataparallel` on SageMaker with PyTorch.

**distribution (dict)(optional):** A dictionary with information on how to run distributed training (default: None).

- To uses `smdistributed.dataparallel` as a distribution strategy, use the following option:

```
distribution={ "smdistributed": { "dataparallel": { "enabled": True } } }
```

- **`custom_mpi_options (str)(optional)`**: Custom MPI options. The following is an example of how you can use this parameter when defining distribution. To learn more, see [Custom MPI Options \(p. 1793\)](#).

```

distribution = {
    "smdistributed": {
        "dataparallel": {
            "enabled": True,
            "custom_mpi_options": "-verbose -x NCCL_DEBUG=VERSION"
        }
    }
}

```

**`train_instance_type (str)(required)`**: A type of Amazon EC2 instance to use.

- If the `smdistributed.dataparallel` distribution strategy is used, the allowed instance types are: `ml.p4d.24xlarge`, `ml.p3dn.24xlarge`, and `ml.p3.16xlarge`. For best performance, it is recommended that you use an EFA supported instance, `ml.p3dn.24xlarge` or `ml.p4d.24xlarge`, with the latest supported version of PyTorch.

### Example

```

from sagemaker.pytorch import PyTorch
pt_estimator = PyTorch(
    base_job_name="training_job_name_prefix",
    entry_point="pt-train.py",
    role="SageMakerRole",
    # You must set py_version to py36
    py_version="py36",
    framework_version="1.8.1",
    # For training with multi node distributed training, set this count.
    # Example: 2,3,4,..8
    instance_count=2,
    # For training with p3dn instance use - ml.p3dn.24xlarge
    instance_type="ml.p3.16xlarge",
    # Training using smdistributed.dataparallel Distributed Training Framework
    distribution={"smdistributed": {"dataparallel": {"enabled": True}}})
)

pt_estimator.fit("s3://bucket/path/to/training/data")

```

### Note

If you are using SageMaker Python SDK 1.x, you need to use hyperparameters to specify `smdistributed.dataparallel` as the distributed training strategy.

```

from sagemaker.pytorch import PyTorch
pt_estimator = PyTorch(
    ...
    # Training using smdistributed.dataparallel Distributed
    Training Framework

    hyperparameters={"sagemaker_distributed_dataparallel_enabled": True},
    ...
)
pt_estimator.fit("s3://bucket/path/to/training/data")

```

# Modify Your Training Script using the SageMaker Data Parallel Library

## Script Modification Overview

SageMaker's distributed data parallel library (the library) APIs are designed for ease of use, and to provide seamless integration with existing distributed training toolkits.

- *SageMaker Python SDK with the library API:* In most cases, all you have to change in your training script is the Horovod or other data parallel library import statements. Swap these out with the SageMaker data parallel library equivalents.
- *Focus on your model training without infrastructure management:* When training a deep learning model with the library on SageMaker, you can focus on your model training, while SageMaker does cluster management: brings up the nodes and creates the cluster, completes the training, then tears down the cluster.

To customize your own training script, you need to do the following:

- You must provide TensorFlow/PyTorch training scripts that are adapted to use the library. The following sections provide example code for this.
- Your input data must be in an S3 bucket or in FSx in the Amazon Region that you will use to launch your training job. If you use the Jupyter Notebooks provided, create a SageMaker notebook instance in the same Region as the bucket that contains your input data. For more information about storing your training data, refer to the [SageMaker Python SDK data inputs](#) documentation.

### Tip

Consider using FSx instead of Amazon S3 to increase training performance. It has higher throughput and lower latency than Amazon S3.

Use the following sections to see examples of training scripts that can be used to convert your TensorFlow or PyTorch training scripts. Then, use one of the example notebooks for your template to launch a training job. You'll need to swap your training script with the one that came with the notebook and modify any input functions as necessary. Once you have launched a training job, you can monitor it using Amazon CloudWatch.

Then you can see how to deploy your trained model to an endpoint by following one of the example notebooks for deploying a model.

Finally, you can follow an example notebook to test inference on your deployed model.

## Modify a TensorFlow 2.x Training Script Using SageMaker Distributed Data Parallel

The following steps show you how to convert a TensorFlow 2.3.1 or 2.4.1 training script to utilize SageMaker's distributed data parallel library.

The library APIs are designed to be similar to Horovod APIs. Refer to the [SageMaker distributed data parallel TensorFlow API documentation](#) for additional details on each API that the library offers for TensorFlow.

### Note

SageMaker distributed data parallel is adaptable to TensorFlow training scripts composed of `tf` core modules except `tf.keras` modules. SageMaker distributed data parallel does not support TensorFlow with Keras implementation.

1. Import the library's TensorFlow client and initialize it:

```
import smdistributed.dataparallel.tensorflow as sdp
sdp.init()
```

2. Pin each GPU to a single `smdistributed.dataparallel` process with `local_rank`—this refers to the relative rank of the process within a given node. The `sdp.tensorflow.local_rank()` API provides you the local rank of the device. The leader node is rank 0, and the worker nodes are rank 1, 2, 3, and so on. This is invoked in the next code block as `sdp.local_rank()`. `set_memory_growth` is not directly related to SageMaker distributed, but must be set for distributed training with TensorFlow.

```
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
if gpus:
    tf.config.experimental.set_visible_devices(gpus[sdp.local_rank()], 'GPU')
```

3. Scale the learning rate by the number of workers. The `sdp.tensorflow.size()` API provides you the number of workers in the cluster. This is invoked in the next code block as `sdp.size()`.

```
learning_rate = learning_rate * sdp.size()
```

4. Use the library's `DistributedGradientTape` to optimize AllReduce operations during training. This wraps `tf.GradientTape`.

```
with tf.GradientTape() as tape:
    output = model(input)
    loss_value = loss(label, output)

# SageMaker data parallel: Wrap tf.GradientTape with the library's
# DistributedGradientTape
tape = sdp.DistributedGradientTape(tape)
```

5. Broadcast the initial model variables from the leader node (rank 0) to all the worker nodes (ranks 1 through n). This is needed to ensure a consistent initialization across all the worker ranks. For this, you use the `sdp.tensorflow.broadcast_variables` API after the model and optimizer variables are initialized. This is invoked in the next code block as `sdp.broadcast_variables()`.

```
sdp.broadcast_variables(model.variables, root_rank=0)
sdp.broadcast_variables(opt.variables(), root_rank=0)
```

6. Finally, modify your script to save checkpoints only on the leader node. The leader node has a synchronized model. This also avoids worker nodes overwriting the checkpoints and possibly corrupting the checkpoints.

```
if sdp.rank() == 0:
    checkpoint.save(checkpoint_dir)
```

The following is an example TensorFlow2 training script for distributed training with the library:

```
import tensorflow as tf

# SageMaker data parallel: Import the library TF API
import smdistributed.dataparallel.tensorflow as sdp

# SageMaker data parallel: Initialize the library
sdp.init()
```

```

gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)
if gpus:
    # SageMaker data parallel: Pin GPUs to a single library process
    tf.config.experimental.set_visible_devices(gpus[sdp.local_rank()], 'GPU')

# Prepare Dataset
dataset = tf.data.Dataset.from_tensor_slices(...)

# Define Model
mnist_model = tf.keras.Sequential(...)
loss = tf.losses.SparseCategoricalCrossentropy()

# SageMaker data parallel: Scale Learning Rate
# LR for 8 node run : 0.000125
# LR for single node run : 0.001
opt = tf.optimizers.Adam(0.000125 * sdp.size())

@tf.function
def training_step(images, labels, first_batch):
    with tf.GradientTape() as tape:
        probs = mnist_model(images, training=True)
        loss_value = loss(labels, probs)

    # SageMaker data parallel: Wrap tf.GradientTape with the library's
    # DistributedGradientTape
    tape = sdp.DistributedGradientTape(tape)

    grads = tape.gradient(loss_value, mnist_model.trainable_variables)
    opt.apply_gradients(zip(grads, mnist_model.trainable_variables))

    if first_batch:
        # SageMaker data parallel: Broadcast model and optimizer variables
        sdp.broadcast_variables(mnist_model.variables, root_rank=0)
        sdp.broadcast_variables(opt.variables(), root_rank=0)

    return loss_value

...
# SageMaker data parallel: Save checkpoints only from master node.
if sdp.rank() == 0:
    checkpoint.save(checkpoint_dir)

```

For more advanced usage, refer to [SageMaker Distributed Data Parallel TensorFlow API documentation](#).

## Modify a PyTorch Training Script Using SMD Data Parallel

The following steps show you how to convert a PyTorch training script to utilize SageMaker's distributed data parallel library.

The library APIs are designed to be similar to PyTorch Distributed Data Parallel (DDP) APIs. For additional details on each data parallel API offered for PyTorch, see the [SageMaker distributed data parallel PyTorch API documentation](#).

1. Import the library's PyTorch client and initialize it, then import the module for distributed training.

```

import smdistributed.dataparallel.torch.distributed as dist

from smdistributed.dataparallel.torch.parallel.distributed import
    DistributedDataParallel as DDP

```

```
dist.init_process_group()
```

2. After parsing arguments and defining a batch size parameter (for example, `batch_size=args.batch_size`), add a 2-line of code to resize the batch size per worker (GPU). PyTorch's `DataLoader` operation does not automatically handle the batch resizing for distributed training.

```
batch_size // dist.get_world_size()  
batch_size = max(batch_size, 1)
```

3. Pin each GPU to a single SageMaker data parallel library process with `local_rank`—this refers to the relative rank of the process within a given node.

The `smdistributed.dataparallel.torch.get_local_rank()` API provides you the local rank of the device. The leader node is rank 0, and the worker nodes are rank 1, 2, 3, and so on. This is invoked in the next code block as `dist.get_local_rank()`.

```
torch.cuda.set_device(dist.get_local_rank())
```

4. Wrap the PyTorch model with the library's DDP.

```
model = ...  
# Wrap model with the library's DistributedDataParallel  
model = DDP(model)
```

5. Modify the `torch.utils.data.distributed.DistributedSampler` to include the cluster's information. Set `num_replicas` to the total number of GPUs participating in training across all the nodes in the cluster. This is called `world_size`. You can get `world_size` with the `smdistributed.dataparallel.torch.get_world_size()` API. This is invoked in the following code as `dist.get_world_size()`. Also supply the node rank using `smdistributed.dataparallel.torch.get_rank()`. This is invoked as `dist.get_rank()`.

```
train_sampler = DistributedSampler(train_dataset, num_replicas=dist.get_world_size(),  
rank=dist.get_rank())
```

6. Modify your script to save checkpoints only on the leader node. The leader node has a synchronized model. This also avoids worker nodes overwriting the checkpoints and possibly corrupting the checkpoints.

The following is an example PyTorch training script for distributed training with the library:

```
# SageMaker data parallel: Import the library PyTorch API  
import smdistributed.dataparallel.torch.distributed as dist  
  
# SageMaker data parallel: Import the library PyTorch DDP  
from smdistributed.dataparallel.torch.parallel.distributed import DistributedDataParallel  
as DDP  
  
# SageMaker data parallel: Initialize the library  
dist.init_process_group()  
  
class Net(nn.Module):  
    ...  
    # Define model  
  
def train(...):  
    ...  
    # Model training
```

```
def test(...):
    ...
    # Model evaluation

def main():

    # SageMaker data parallel: Scale batch size by world size
    batch_size // dist.get_world_size()
    batch_size = max(batch_size, 1)

    # Prepare dataset
    train_dataset = torchvision.datasets.MNIST(...)

    # SageMaker data parallel: Set num_replicas and rank in DistributedSampler
    train_sampler = torch.utils.data.distributed.DistributedSampler(
        train_dataset,
        num_replicas=dist.get_world_size(),
        rank=dist.get_rank())

    train_loader = torch.utils.data.DataLoader(...)

    # SageMaker data parallel: Wrap the PyTorch model with the library's DDP
    model = DDP(Net().to(device))

    # SageMaker data parallel: Pin each GPU to a single library process.
    torch.cuda.set_device(local_rank)
    model.cuda(local_rank)

    # Train
    optimizer = optim.Adadelta(...)
    scheduler = StepLR(...)
    for epoch in range(1, args.epochs + 1):
        train(...)
        if rank == 0:
            test(...)
        scheduler.step()

    # SageMaker data parallel: Save model on master node.
    if dist.get_rank() == 0:
        torch.save(...)

if __name__ == '__main__':
    main()
```

For more advanced usage, see the [SageMaker distributed data parallel PyTorch API documentation](#).

## Launch a Training Job

To launch the training job using your SageMaker distributed data parallel configured training script, you use the `Estimator.fit()` function. We recommend that you launch a training job using a SageMaker notebook instance or SageMaker Studio. To see an example of how you can launch a training job using a Studio or a notebook instance, see [Distributed Training Jupyter Notebook Examples \(p. 1819\)](#).

Use the following resources to learn more about using the SageMaker Python SDK with these frameworks:

- [Use TensorFlow with the SageMaker Python SDK](#)
- [Using PyTorch with the SageMaker Python SDK](#)

## SageMaker distributed data parallel Configuration Tips and Pitfalls

Review the following tips and pitfalls before using SageMaker's distributed data parallel library. This list includes tips that are applicable across frameworks.

### Topics

- [Data Preprocessing \(p. 1792\)](#)
- [Single vs Multiple Nodes \(p. 1792\)](#)
- [Debug Scaling Efficiency with Debugger \(p. 1792\)](#)
- [Batch Size \(p. 1792\)](#)
- [Custom MPI Options \(p. 1793\)](#)

### Data Preprocessing

If you pre-process data during training using an external library that utilizes the CPU, you may run into a CPU bottleneck because SageMaker distributed data parallel uses the CPU for all-reduce operations. You may be able to improve training time by moving pre-processing steps to a library that uses GPUs or by completing all pre-processing before training.

### Single vs Multiple Nodes

It is recommended that you use this library with multiple nodes. The library can be used with a single-host, multi-device setup (for example, a single ML compute instance with multiple GPUs), however, when you use two or more nodes, the library's AllReduce operation gives you significant performance improvement. Also, on a single host, NVLink already contributes to in-node AllReduce efficiency.

### Debug Scaling Efficiency with Debugger

You can use Amazon SageMaker Debugger to monitor and visualize CPU and GPU utilization, and other metrics of interest, during training. You can use Debugger [built-in rules](#) to monitor computational performance issues, such as CPUBottleneck, LoadBalancing, and LowGPUUtilization. You can specify these rules with [Debugger configurations](#) when you define an Amazon SageMaker Python SDK estimator. If you use Amazon CLI and Amazon Boto3 for training on SageMaker, you can enable Debugger as shown in [Configure Debugger Using Amazon SageMaker API](#).

To see an example using Debugger in a SageMaker training job, you can reference one of the notebook example in the [SageMaker Notebook Examples GitHub repository](#). To learn more about Debugger, see [Amazon SageMaker Debugger](#).

### Batch Size

In distributed training, as more nodes are added, batch sizes should increase proportionally. To improve convergence speed as you add more nodes to your training job and increase the global batch size, increase the learning rate.

One way to achieve this is by using a gradual learning rate warmup where the learning rate is ramped up from a small to a large value as the training job progresses. This ramp avoids a sudden increase of the learning rate, allowing healthy convergence at the start of training. For example, you can use a "Linear Scaling Rule" where each time the mini-batch size is multiplied by k, the learning rate is also multiplied by k. To learn more about this technique, see the research paper, Accurate, [Large Minibatch SGD: Training ImageNet in 1 Hour](#), Sections 2 and 3.

## Custom MPI Options

The SageMaker distributed data parallel library employs Message Passing Interface (MPI), a popular standard for managing communication between nodes in a high-performance cluster, and uses NVIDIA's NCCL library for GPU-level communication. When you use the data parallel library with a TensorFlow or Pytorch Estimator, the respective container sets up the MPI environment and executes the `mpirun` command to start jobs on the cluster nodes.

You can set custom MPI operations using the `custom_mpi_options` parameter in the `Estimator`. Any `mpirun` flags passed in this field are added to the `mpirun` command and executed by Amazon SageMaker for training. For example, you may define the `distribution` parameter of an `Estimator` using the following to use the `NCCL_DEBUG` variable to print the NCCL version at the start of the program:

```
distribution = {'smdistributed': {'dataparallel': {'enabled': True, "custom_mpi_options": "-verbose -x NCCL_DEBUG=VERSION"}}}
```

## Data Parallel Library FAQ

Use the following to find answers to commonly asked questions about SageMaker's data parallelism library.

**Q: When using the library, how are the allreduce-supporting CPU instances managed? Do I have to create heterogeneous CPU-GPU clusters, or does the SageMaker service create extra C5s for jobs that use the library?**

The library uses the CPUs available with GPU instances. No additional C5 or CPU instances are launched; if your SageMaker training job is 8 node `m1.p3dn.24xlarge` clusters, only 8 `m1.p3dn.24xlarge` instances are used. No additional instances are provisioned.

**Q: I have a training job taking 5 days on a single `m1.p3.24xlarge` instance with a set of hyperparameters H1 (learning rate, batch size, optimizer, etc). Is enabling SDP and using a 5x bigger cluster enough to experience an approximate 5x speedup? Or will I have to revisit its training hyperparameters after enabling SDP?**

Enabling the library would change the overall batch size. The new overall batch size is scaled linearly with the number of training instances used. As a result of this, hyperparameters, such as learning rate, have to be changed to ensure convergence.

**Q: Does the library support Spot?**

Yes. You can use managed spot training. You specify the path to the checkpoint file in the SageMaker training job. You enable save/restore checkpoints in their training script as mentioned in the last step of the section called "Script Modification Overview" (p. 1787).

**Q: What is the difference between the library's balanced fusion buffers and PyTorch Distributed Data Parallel (DDP) "Gradient Buckets"?**

The primary difference is that DDP's fusion buffer is used for AllReduce and the library's balanced fusion buffer is used for parameter server style synchronization. The library is the first framework to shard fusion buffers in parameter server-based gradient synchronization.

From the [PyTorch DDP documentation](#): "To improve communication efficiency, the Reducer organizes parameter gradients into buckets, and reduces one bucket at a time. Bucket size can be configured by setting the `bucket_cap_mb` argument in DDP constructor. The mapping from parameter gradients to buckets is determined at the construction time, based on the bucket size limit and parameter sizes. Model parameters are allocated into buckets in (roughly) the reverse order of `Model.parameters()`

from the given model. The reason for using the reverse order is because DDP expects gradients to become ready during the backward pass in approximately that order.”

**Q: Is the library relevant in single-host, multi-device setup?**

The library can be used with a single-host, multi-device setup. However, in two or more nodes, the library’s AllReduce operation gives you significant performance improvement. Also, on a single host, NVLink already contributes to in-node AllReduce efficiency.

**Q: Can the library be used with PyTorch Lightning?**

No. However, with the library’s DDP for PyTorch, you can write custom DDP to achieve the functionality.

**Q: Where should the training dataset be stored?**

The training dataset can be stored in an S3 bucket or on an FSx drive. See this [document for various supported input filesystem for a training job](#).

**Q: When using the library, is it mandatory to have training data in FSx for Lustre? Can EFS and S3 be used?**

We recommend using FSx to cut down the time to kickstart the training. It is not mandatory.

**Q: Can the library be used with CPU nodes?**

No. The library supports m1.p3.16xlarge, m1.p3dn.24xlarge, and m1.p4d.24xlarge instances at this time.

**Q: What frameworks and framework versions are currently supported by the library at launch?**

The library currently supports PyTorch v1.6.0 or later and Tensorflow v2.3.0 or later. It doesn’t support Tensorflow 1.x. For more information about which version of the library is packaged within Amazon deep learning containers, see [Release Notes for Deep Learning Containers](#).

## Data Parallel Troubleshooting

If you run into an error, you can use the following list to try to troubleshoot your training job. If the problem persists, contact Amazon Support.

**Topics**

- [Considerations for Using SageMaker Distributed Data Parallel with SageMaker Debugger and Checkpoints \(p. 1794\)](#)
- [An Unexpected Prefix \(model for example\) Is Attached to state\\_dict keys \(model parameters\) from a PyTorch Distributed Training Job \(p. 1795\)](#)

### Considerations for Using SageMaker Distributed Data Parallel with SageMaker Debugger and Checkpoints

To monitor system bottlenecks, profile framework operations, and debug model output tensors for training jobs with SageMaker distributed data parallel, you use SageMaker Debugger.

However, when you use SageMaker Debugger, SageMaker distributed data parallel, and SageMaker checkpoints, you might see an error that looks like the following:

SMD-debug Does Not Currently Support Distributed Training Jobs With Checkpointing Enabled

This is due to an internal error between Debugger and checkpoints, which occurs when you enable SageMaker Distributed.

- If you enable all three features, SageMaker Python SDK automatically turns off Debugger by passing `debugger_hook_config=False`, which is equivalent to the following framework estimator example.

```
bucket=sagemaker.Session().default_bucket()
base_job_name="sagemaker-checkpoint-test"
checkpoint_in_bucket="checkpoints"

# The S3 URI to store the checkpoints
checkpoint_s3_bucket="s3://{}//{}/{}".format(bucket, base_job_name, checkpoint_in_bucket)

estimator = TensorFlow(
    ...
    distribution={"smdistributed": {"dataparallel": { "enabled": True }}},
    checkpoint_s3_uri=checkpoint_s3_bucket,
    checkpoint_local_path="/opt/ml/checkpoints",
    debugger_hook_config=False
)
```

- If you want to keep using both SageMaker distributed data parallel and SageMaker Debugger, a workaround is manually adding checkpointing functions to your training script instead of specifying the `checkpoint_s3_uri` and `checkpoint_local_path` parameters from the estimator. For more information about setting up manual checkpointing in a training script, see [Saving Checkpoints \(p. 1818\)](#).

## An Unexpected Prefix (`model` for example) Is Attached to `state_dict` keys (model parameters) from a PyTorch Distributed Training Job

The SageMaker data parallel library does not directly alter or prepend any model parameter names when PyTorch training jobs save model artifacts. The PyTorch's distributed training changes the names in the `state_dict` to go over the network, prepending the prefix. If you encounter any model failure problem due to the different parameter names while you are using the SageMaker data parallel library and checkpointing for PyTorch training, adapt the following example code to remove the prefix at the step you load checkpoints in your training script:

```
state_dict = {k.partition('model.')[2]:state_dict[k] for k in state_dict.keys()}
```

This takes each `state_dict` key as a string value, separates the string at the first occurrence of '`model.`', and takes the third list item (with index 2) of the partitioned string.

For more information about the prefix issue, see a discussion thread at [Prefix parameter names in saved model if trained by multi-GPU?](#) in the *PyTorch discussion forum*.

For more information about the PyTorch methods for saving and loading models, see [Saving & Loading Model Across Devices](#) in the *PyTorch documentation*.

## SageMaker's Distributed Model Parallel

### Important

To use new features with an existing notebook instance or Studio app, restart it to get the latest updates.

Amazon SageMaker's distributed model parallel library (the library) can be used to train large deep learning models that were previously difficult to train due to GPU memory limitations. The library automatically and efficiently splits a model across multiple GPUs and instances and coordinates model training, allowing you to increase prediction accuracy by creating larger models with more parameters.

You can use the library to automatically partition your existing TensorFlow and PyTorch workloads across multiple GPUs with minimal code changes. You can access the library's API through the SageMaker SDK.

Use the following sections to learn more about model parallelism and the SageMaker model parallel library. This library's API documentation is located in the SageMaker Python SDK under [Distributed Training APIs](#). To see the latest updates to the library, refer to the [release notes](#).

### Topics

- [Introduction to Model Parallelism \(p. 1796\)](#)
- [Core Features of SageMaker Distributed Model Parallel \(p. 1797\)](#)
- [Use the SageMaker Distributed Model Parallel API \(p. 1801\)](#)
- [Modify Your Training Script Using SageMaker's Distributed Model Parallel Library \(p. 1804\)](#)
- [SageMaker distributed model parallel Configuration Tips and Pitfalls \(p. 1816\)](#)
- [Model Parallel Troubleshooting \(p. 1817\)](#)

## Introduction to Model Parallelism

**Model parallelism** is the process of splitting a model up between multiple devices or nodes (such as GPU-equipped instances) and creating an efficient pipeline to train the model across these devices to maximize GPU utilization.

Use the sections on this page to learn more about model parallelism, including how it can be used to overcome issues that arise when training large ML models, and important considerations when integrating it into your ML training script.

### What is Model Parallelism?

Increasing deep learning model size (layers and parameters) can result in better accuracy, however there is a limit to the maximum model size you can fit in a single GPU. When training deep learning models, GPU memory limitations can be a bottleneck in the following ways:

- They can limit the size of the model you train. Given that larger models tend to achieve higher accuracy, this directly translates to trained model accuracy.
- They can limit the batch size you train with, leading to lower GPU utilization and slower training.

To overcome the limitations associated with training a model on a single GPU, you can use model parallelism to distribute and train your model on multiple computing devices.

### Important Considerations when Using Model Parallelism

When you use model parallelism, you must consider the following:

- **How you split your model across devices** : The computational graph of your model, sizes of model parameters and activations, and your resource constraints (for example, time vs. memory) determine the best partitioning strategy.

To reduce the time and effort required to efficiently split your model, you can use automated model splitting features offered by Amazon SageMaker's distributed model parallel library.

- **Achieving parallelization** : Model training—that is, forward computations and backward propagation—is inherently sequential, where each operation must wait for its inputs to be computed by another operation. For this reason, forward and backward pass stages of deep learning training are not easily parallelizable, and naively splitting a model across multiple GPUs may lead to poor device utilization. For example, a layer on GPU  $i+1$  has to wait for the output from a layer on GPU  $i$ , and so GPU  $i+1$  remains idle during this waiting period.

The model parallel library can achieve true parallelization by implementing pipelined execution by building an efficient computation schedule where different devices can work on forward and backward passes for different data samples at the same time.

To learn how you can use the library to efficiently split your model across devices and improve device utilization during training, see [Core Features of SageMaker Distributed Model Parallel \(p. 1797\)](#).

## Core Features of SageMaker Distributed Model Parallel

Amazon SageMaker's distributed model parallel makes model parallelism more accessible by providing automated model splitting and sophisticated pipeline execution scheduling. The model splitting algorithms can optimize for speed or memory consumption. The library also supports manual partitioning. When you use the library, training is executed in a pipelined fashion over microbatches to maximize GPU usage.

You can configure these features using a few lines of code when you create your training script and define your SageMaker PyTorch or Tensorflow Estimator. Use the following sections to learn more about these core features of the library.

### Automated Model Splitting

When you use SageMaker's model parallel library, you can take advantage of *automated model splitting*, also referred to as *automated model partitioning*. The library uses a partitioning algorithm that balances memory, minimizes communication between devices, and optimizes performance. You can configure the automated partitioning algorithm to optimize for speed or memory.

Alternatively, you can use manual model splitting. We recommend automated model splitting, unless you are very familiar with the model architecture and have a good idea of how to efficiently partition your model.

### How It Works

Auto-partitioning occurs during the first training step, when the `smp.step`-decorated function is first called. During this call, the library first constructs a version of the model on the CPU RAM (to avoid GPU memory limitations), and then analyzes the model graph and makes a partitioning decision. Based on this decision, each model partition is loaded on a GPU, and only then the first step is executed. Because of these analysis and partitioning steps, the first training step might take longer.

In either framework, the library manages the communication between devices through its own backend, which is optimized for Amazon infrastructure.

The auto-partition design adapts to the characteristics of the framework, and the library does the partitioning at the granularity level that is more natural in each framework. For instance, in TensorFlow, each specific operation can be assigned to a different device, whereas in PyTorch, the assignment is done at the module level, where each module consists of multiple operations. The follow section reviews the specifics of the design in each framework.

### Automated Model Splitting with PyTorch

During the first training step, the model parallel library internally runs a tracing step that is meant to construct the model graph and determine the tensor and parameter shapes. After this tracing step, the library constructs a tree, which consists of the nested `nn.Module` objects in the model, as well as additional data gathered from tracing, such as the amount of stored `nn.Parameters`, and execution time for each `nn.Module`.

Next, the library traverses this tree from the root and runs a partitioning algorithm that assigns each `nn.Module` to a device, which balances computational load (measured by module execution time) and memory use (measured by the total stored `nn.Parameter` size and activations). If multiple `nn.Modules`

share the same `nn.Parameter`, then these modules are placed on the same device to avoid maintaining multiple versions of the same parameter. Once the partitioning decision is made, the assigned modules and weights are loaded to their devices.

### Automated Model Splitting with TensorFlow

The model parallel library analyzes the sizes of the trainable variables and the graph structure, and internally uses a graph partitioning algorithm. This algorithm comes up with a device assignment for each operation, with the objective of minimizing the amount of communication needed across devices, subject to two constraints:

- Balancing the number of variables stored in each device
- Balancing the number of operations executed in each device

If you specify `speed` for `optimize` (in the model parallel parameters in the Python SDK), the library tries to balance the number of operations and `tf.Variable` objects in each device. Otherwise, it tries to balance the total size of `tf.Variables`.

Once the partitioning decision is made, the library creates a serialized representation of the subgraph that each device needs to execute and imports them onto each device. While partitioning, the library places operations that consume the same `tf.Variable` and operations that are part of the same Keras layer onto the same device. It also respects the colocation constraints imposed by TensorFlow. This means that, for example, if there are two Keras layers that share a `tf.Variable`, then all operations that are part of these layers are placed on a single device.

### Comparison of Automated Model Splitting Between Frameworks

In TensorFlow, the fundamental unit of computation is a `tf.Operation`, and TensorFlow represents the model as a directed acyclic graph (DAG) of `tf.Operations`, and therefore model parallel library partitions this DAG so that each node goes to one device. Crucially, `tf.Operation` objects are sufficiently rich with customizable attributes, and they are universal in the sense that every model is guaranteed to consist of a graph of such objects.

PyTorch on the other hand, does not have an equivalent notion of operation that is sufficiently rich and universal. The closest unit of computation in PyTorch that has these characteristics is an `nn.Module`, which is at a much higher granularity level, and this is why the library does partitioning at this level in PyTorch.

### Manual Model Splitting

If you want to manually specify how your model is partitioned across devices, you can use manual model splitting by using `smp.partition` context managers.

To use this option, set `auto_partition` to `False`, and define a `default_partition` in the SageMaker Python SDK. Any operation that is not explicitly placed on a partition through the `smp.partition` context manager is executed on the `default_partition`. In this case, the automated splitting logic is bypassed, and each operation is placed based on your specification. Based on the resulting graph structure, the model parallel library creates a pipelined execution schedule automatically.

### Pipeline Execution Schedule

A core feature of SageMaker's distributed model parallel library is *pipelined execution*, which determines the order in which computations are made and data is processed across devices during model training. Pipelining is a technique to achieve true parallelization in model parallelism, by having the GPUs compute simultaneously on different data samples, and to overcome the performance loss due to sequential computation.

Pipelining is based on splitting a mini-batch into microbatches, which are fed into the training pipeline one-by-one and follow an execution schedule defined by the library runtime. A *microbatch* is a smaller

subset of a given training mini-batch. The pipeline schedule determines which microbatch is executed by which device for every time slot.

For example, depending on the pipeline schedule and the model partition, GPU  $i$  might perform (forward or backward) computation on microbatch  $b$  while GPU  $i+1$  performs computation on microbatch  $b+1$ , thereby keeping both GPUs active at the same time. During a single forward or backward pass, execution flow for a single microbatch might visit the same device multiple times, depending on the partitioning decision. For instance, an operation that is at the beginning of the model might be placed on the same device as an operation at the end of the model, while the operations in between are on different devices, which means this device is visited twice.

The library offers two different pipeline schedules, *simple* and *interleaved*, which can be configured using the `pipeline` parameter in the SageMaker Python SDK. In most cases, interleaved pipeline can achieve better performance by utilizing the GPUs more efficiently.

### Interleaved Pipeline

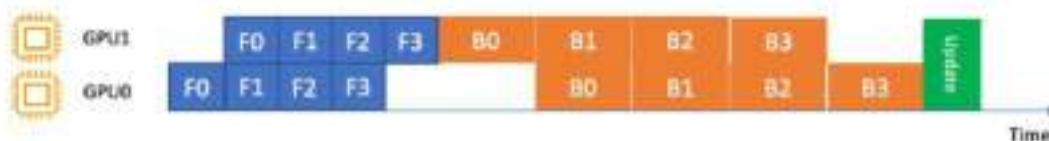
In an interleaved pipeline, backward execution of the microbatches is prioritized whenever possible. This allows quicker release of the memory used for activations, using memory more efficiently. It also allows for scaling the number of microbatches higher, reducing the idle time of the GPUs. At steady-state, each device alternates between running forward and backward passes. This means that the backward pass of one microbatch may run before the forward pass of another microbatch finishes.



The preceding figure illustrates an example execution schedule for the interleaved pipeline over 2 GPUs. In the figure, F0 represents the forward pass for microbatch 0, and B1 represents the backward pass for microbatch 1. **Update** represents the optimizer update of the parameters. GPU0 always prioritizes backward passes whenever possible (for instance, executes B0 before F2), which allows for clearing of the memory used for activations earlier.

### Simple Pipeline

A simple pipeline, by contrast, finishes running the forward pass for each microbatch before starting the backward pass. This means that it only pipelines the forward pass and backward pass stages within themselves. The following figure illustrates an example of how this works, over 2 GPUs.

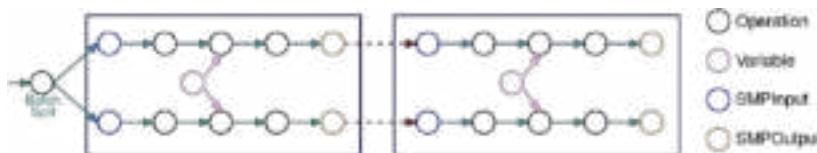


### Pipelining Execution in Specific Frameworks

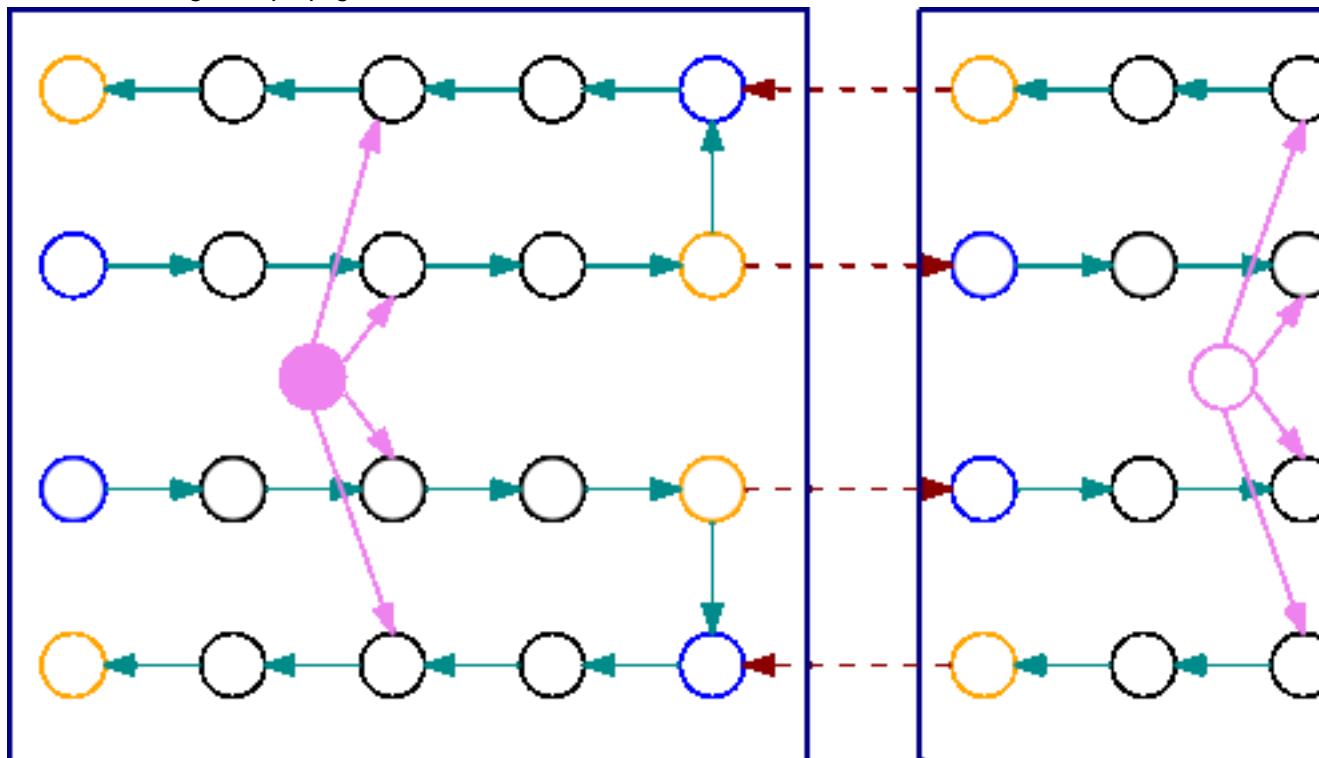
Use the following sections to learn about the framework-specific pipeline scheduling decisions SageMaker's distributed model parallel library makes for Tensorflow and PyTorch.

#### Pipeline Execution with TensorFlow

The following image is an example of a TensorFlow graph partitioned by the model parallel library, using automated model splitting. When a graph is split, each resulting subgraph is replicated  $B$  times (except for the variables), where  $B$  is the number of microbatches. In this figure, each subgraph is replicated 2 times ( $B=2$ ). An `SMPInput` operation is inserted at each input of a subgraph, and an `SMPOutput` operation is inserted at each output. These operations communicate with the library backend to transfer tensors to and from each other.



The following image is an example of 2 subgraphs split with  $B=2$  with gradient operations added. The gradient of a **SMPInput** op is a **SMPOutput** op, and vice versa. This enables the gradients to flow backwards during back-propagation.



This GIF demonstrates an example interleaved pipeline execution schedule with  $B=2$  microbatches and 2 subgraphs. Each device sequentially executes one of the subgraph replicas to improve GPU utilization. As  $B$  grows larger, the fraction of idle time slots goes to zero. Whenever it is time to do (forward or backward) computation on a specific subgraph replica, the pipeline layer signals to the corresponding blue **SMPInput** operations to start executing.

Once the gradients from all microbatches in a single mini-batch are computed, the library combines the gradients across microbatches, which can then be applied to the parameters.

### Pipeline Execution with PyTorch

Conceptually, pipelining follows a similar idea in PyTorch. However, since PyTorch does not involve static graphs and so the model parallel library's PyTorch feature uses a more dynamic pipelining paradigm.

As in TensorFlow, each batch is split into a number of microbatches, which are executed one at a time on each device. However, the execution schedule is handled via execution servers launched on each device. Whenever the output of a submodule that is placed on another device is needed on the current device, an execution request is sent to the execution server of the remote device along with the input tensors to the submodule. The server then executes this module with the given inputs and returns the response to the current device.

Since the current device is idle during the remote submodule execution, the local execution for the current microbatch pauses, and the library runtime switches execution to another microbatch which the current device can actively work on. The prioritization of microbatches is determined by the chosen pipeline schedule. For an interleaved pipeline schedule, microbatches that are in the backward stage of the computation are prioritized whenever possible.

## Use the SageMaker Distributed Model Parallel API

To use Amazon SageMaker's distributed model parallel library, you must create a training script for one of the supported frameworks and launch the training job using the SageMaker Python SDK. To learn how you can incorporate the library into a training script, see [Modify Your Training Script Using SageMaker's Distributed Model Parallel Library \(p. 1804\)](#). The library's API documentation is located in the SageMaker Python SDK. Refer to the [SageMaker's distributed model parallel API documentation](#).

SageMaker supports the following training environment configurations.

1. You can use a prebuilt TensorFlow or PyTorch container. This option is recommended for new users of the model parallel library, and is demonstrated in the example [MNIST with PyTorch 1.6 and Amazon SageMaker's distributed model parallel library](#).
2. You can extend SageMaker prebuilt containers to handle any additional functional requirements for your algorithm or model that the prebuilt SageMaker Docker image doesn't support. To see an example of how you can extend a pre-built container, see [Extend a Prebuilt Container](#).
3. You can adapt your own Docker container to work with SageMaker using the [SageMaker Training toolkit](#). For an example, see [Adapting Your Own Training Container](#).

For options 2 and 3 in the preceding list, refer to [Extend or Adapt A Docker Container that Contains SageMaker's Distributed Model Parallel Library \(p. 1803\)](#) to learn how to install the model parallel library in an extended or customized Docker container.

In all cases, you launch your job using a SageMaker Python SDK TensorFlow or PyTorch `Estimator` to initialize the library and launch a training job. See the following section, [Launch a Training Job with the SageMaker Python SDK \(p. 1801\)](#), to learn more.

**Important**

If you launch a training job using an `ml.p4d` instance type (such as `ml.p4d.24xlarge`), for best performance set the following flags in the `mpirun` command. If you use one of these flags, you *must* use the other.

```
-x FI_EFA_USE_DEVICE_RDMA=1 -x FI_PROVIDER=efa
```

Additionally, if you are using PyTorch, you must set the data loader variable `num_workers=0`. To Learn more about the data loader requirement, see [Important Considerations \(p. 1810\)](#).

## Launch a Training Job with the SageMaker Python SDK

The SageMaker Python SDK supports managed training of TensorFlow and PyTorch models. To launch a training job using one of these frameworks, you can define a `TensorFlow Estimator` or a `PyTorch Estimator`.

The TensorFlow and PyTorch `Estimator` object contains a `distribution` parameter, which is used to enable and specify parameters for the initialization of SageMaker's distributed model parallel library. The library internally uses MPI, so in order to use model parallelism, MPI must be enabled using the `distribution` parameter.

The following is an example of how you can launch a new PyTorch training job with the library.

```
sagemaker_session = sagemaker.session.Session(boto_session=session)

mpi_options = {
    "enabled" : True,
    "processes_per_host" : 8,
    "custom_mpi_options" : "--mca btl_vader_single_copy_mechanism none "
}

smp_options = {
    "enabled":True,
    "parameters": {
        "microbatches": 4,
        "placement_strategy": "spread",
        "pipeline": "interleaved",
        "optimize": "speed",
        "partitions": 2,
        "ddp": True,
    }
}

smd_mp_estimator = PyTorch(
    entry_point="pt_mnist.py", # Pick your train script
    source_dir='utils',
    role=role,
    instance_type='ml.p3.16xlarge',
    sagemaker_session=sagemaker_session,
    framework_version='1.6.0',
    # You must set py_version to py36
    py_version='py36',
    instance_count=1,
    distribution={
        "smdistributed": {"modelparallel": smp_options},
        "mpi": mpi_options
    },
    base_job_name="SMD-MP-demo",
)
smd_mp_estimator.fit('s3://my_bucket/my_training_data/')
```

To enable the library, a dictionary with the keys "`mpi`" and "`smdistributed`" needs to be passed as the `distribution` argument of the TensorFlow and PyTorch `Estimator` constructors in Python SDK. For the "`mpi`" key, a dict must be passed which contains:

- "`enabled`": `True` to launch the training job with MPI.
- "`processes_per_hostAmazon EC2 ml instance. The SageMaker Python SDK maintains a one-to-one mapping between processes and GPUs across model and data parallelism. This means that SageMaker schedules each process on a single, separate GPU and no GPU contains more than one process. If you are using PyTorch, you must restrict each process to its own device through torch.cuda.set_device(smp.local_rank()). To learn more, see PyTorch \(p. 1813\).`

### **Important**

`process_per_host` must be less than the number of GPUs per instance and typically will be equal to the number of GPUs per instance.

For example, if you use one instance with 4-way model parallelism and 2-way data parallelism, then `processes_per_host` should be  $2 \times 4 = 8$ . Therefore, you must choose an instance that has at least 8 GPUs, such as an `ml.p3.16xlarge`.

The following image illustrates how 2-way data parallelism and 4-way model parallelism is distributed across 8 GPUs: the models is partitioned across 4 GPUs, and each partition is added to 2 GPUs.

- "custom\_mpi\_options": Use this key to pass any custom MPI options you might need. To avoid Docker warnings from contaminating your training logs, we recommend the following flag.

```
--mca btl_vader_single_copy_mechanism none
```

If you launch a training job using an `ml.p4d` instance type (such as `ml.p4d.24xlarge`), for best performance use the following flags in the `mpirun` command:

```
-x FI_EFA_USE_DEVICE_RDMA=1 -x FI_PROVIDER=efa
```

For the "smdistributed" key, a dictionary must be passed which has the only key "modelparallel". Using "modelparallel" and "dataparallel" in the same training job is not supported.

For the "modelparallel" dictionary, an inner dictionary must be passed, which enables the `modelparallel` library by setting "enabled": `True`, as well as a dict of "parameters" to customize the library. Among the parameters, the "partitions" key is required, which specifies how many model partitions are requested. To learn more about values you can use in `parameters`, see the [SageMaker distributed data parallel API documentation](#).

To launch the training job using SageMaker model parallel configured training script, you use the `Estimator.fit()` function. You can launch a SageMaker training job using a SageMaker notebook instance, or locally.

- Using a SageMaker notebook instance is recommended for new users. To see an example of how you can launch a training job using a SageMaker notebook instance, see [Distributed Training Jupyter Notebook Examples \(p. 1819\)](#).
- You can launch locally if you have [set up your Amazon credentials and Region for development](#).

Use the following resources to learn more about using the SageMaker Python SDK with these frameworks:

- [Use TensorFlow with the SageMaker Python SDK](#)
- [Use PyTorch with the SageMaker Python SDK](#)

## Extend or Adapt A Docker Container that Contains SageMaker's Distributed Model Parallel Library

To extend a pre-built container, or adapt your own container to use SageMaker's distributed model parallel library, you must use one of the PyTorch or TensorFlow GPU general framework base-images. The distributed model parallel library is included in all CUDA 11 (`cu11x`) TensorFlow 2.3.x and PyTorch 1.6.x versioned images and later. See [Available Deep Learning Containers Images](#) for a list of available images.

### Tip

It is recommended that you use the image that contains the latest version of TensorFlow or PyTorch to access the most up to date version of the SageMaker distributed model parallel library.

For example, if you are using PyTorch 1.8.1, your Dockerfile should contain a `FROM` statement similar to the following:

```
# SageMaker PyTorch image
FROM 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-training:1.8.1-gpu-py36-cu111-
ubuntu18.04

# Add your dependencies here
RUN ...

ENV PATH="/opt/ml/code:${PATH}"

# this environment variable is used by the SageMaker PyTorch container to determine our
# user code directory.
ENV SAGEMAKER_SUBMIT_DIRECTORY /opt/ml/code

# /opt/ml and all subdirectories are utilized by SageMaker, use the /code subdirectory to
# store your user code.
COPY cifar10.py /opt/ml/code/cifar10.py
```

Additionally, when you define a PyTorch or TensorFlow estimator, you must specify that the `entry_point` for your training script. This should be the same path identified with `ENV SAGEMAKER_SUBMIT_DIRECTORY` in your Dockerfile.

You must push this docker container to Amazon Elastic Container Registry (Amazon ECR) and use the image URI (`image_uri`) to define an estimator. See [Extend a Prebuilt Container](#) for more information.

For example, if your Dockerfile was defined using the preceding code block and used `name` and `tag` to push it to Amazon ECR, you would define a PyTorch estimator as follows. This example assumes you have already defined `smp_options` and `mpi_options`.

```
smd_mp_estimator = PyTorch(
    entry_point='/opt/ml/code/pt_mnist.py', # Identify
    source_dir='utils',
    role=role,
    instance_type='ml.p3.16xlarge',
    sagemaker_session=sagemaker_session,
    image_uri='aws_account_id.dkr.ecr.region.amazonaws.com/name:tag'
    instance_count=1,
    distribution={
        "smdistributed": smp_options,
        "mpi": mpi_options
    },
    base_job_name="SMD-MP-demo",
)
smd_mp_estimator.fit('s3://my_bucket/my_training_data/')
```

## Modify Your Training Script Using SageMaker's Distributed Model Parallel Library

Use this section to learn how to customize your training script with Amazon SageMaker's distributed model parallel library-specific API functions and parameters. We recommend you use this documentation alongside the [model parallel library API documentation](#).

The training script examples provided in these sections are simplified and designed to highlight the required changes you must make to use the library. For end to end, runnable notebook examples that demonstrate how to use a TensorFlow or PyTorch training script with the SageMaker distributed model parallel library, see [Distributed Training Jupyter Notebook Examples \(p. 1819\)](#).

## Topics

- [Modify a TensorFlow Training Script \(p. 1805\)](#)
- [Modify a PyTorch Training Script \(p. 1810\)](#)

## Modify a TensorFlow Training Script

The following are examples of training scripts that you can use to configure the SageMaker distributed model parallel library with TensorFlow versions 2.3.1 and 2.4.1 with auto-partitioning and manual partitioning. This selection of examples also includes an example integrated with Horovod for hybrid model and data parallelism. We recommend that you review [Unsupported Framework Features \(p. 1805\)](#) before creating a training script.

The required modifications you must make to your training script to use the library are listed in [TensorFlow \(p. 1805\)](#).

To learn how to modify your training script to use hybrid model and data parallelism with Horovod, see [TensorFlow with Horovod \(p. 1807\)](#).

If you want to use manual partitioning, also review [Manual partitioning with TensorFlow \(p. 1809\)](#).

### Tip

For end to end, runnable notebook examples that demonstrate how to use a TensorFlow training script with the SageMaker distributed model parallel library, see [TensorFlow Examples \(p. 1820\)](#).

Note that auto-partitioning is enabled by default. Unless otherwise specified, the following example scripts use auto-partitioning.

## Topics

- [Unsupported Framework Features \(p. 1805\)](#)
- [TensorFlow \(p. 1805\)](#)
- [TensorFlow with Horovod \(p. 1807\)](#)
- [Manual partitioning with TensorFlow \(p. 1809\)](#)

## Unsupported Framework Features

The following TensorFlow features are unsupported by the library:

- `tf.GradientTape()` is currently not supported. You can use `Optimizer.get_gradients()` or `Optimizer.compute_gradients()` instead to compute gradients.
- The `tf.train.Checkpoint.restore()` API is currently not supported. For checkpointing, use `smp.CheckpointManager` instead, which provides the same API and functionality. Note that checkpoint restores with `smp.CheckpointManager` should take place after the first step.

## TensorFlow

The following training script changes are required to run a TensorFlow model with SageMaker's distributed model parallel library:

1. Import and initialize the library with `smp.init()`.
2. Define Keras model by inheriting from `smp.DistributedModel` instead of Keras Model class. Return the model outputs from the call method of the `smp.DistributedModel` object. Be mindful that any tensors returned from the call method will be broadcast across model-parallel devices, incurring communication overhead, so any tensors that are not needed outside the call method (such as intermediate activations) should not be returned.

3. Set `drop_remainder=True` in `tf.Dataset.batch()` method. This is to ensure that the batch size is always divisible by the number of microbatches.
4. Seed the random operations in the data pipeline using `smp.dp_rank()`, e.g., `shuffle(ds, seed=smp.dp_rank())` to ensure consistency of data samples across GPUs that hold different model partitions.
5. Put the forward and backward logic in a step function and decorate it with `smp.step`.
6. Perform post-processing on the outputs across microbatches using `StepOutput` methods such as `reduce_mean`. The `smp.step` function must have a return value that depends on the output of `smp.DistributedModel`.
7. If there is an evaluation step, similarly place the forward logic inside an `smp.step`-decorated function and post-process the outputs using `StepOutput API`.

To learn more about the SageMaker's distributed model parallel library API, refer to the [API documentation](#).

```

import tensorflow as tf

# smdistributed: Import TF2.x API
import smdistributed.modelparallel.tensorflow as smp

# smdistributed: Initialize
smp.init()

# Download and load MNIST dataset.
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data(
    "MNIST-data-%d" % smp.rank()
)
x_train, x_test = x_train / 255.0, x_test / 255.0

# Add a channels dimension
x_train = x_train[..., tf.newaxis]
x_test = x_test[..., tf.newaxis]

# smdistributed: If needed, seed the shuffle with smp.dp_rank(), and drop_remainder # in batching to make sure batch size is always divisible by number of microbatches
train_ds = (
    tf.data.Dataset.from_tensor_slices((x_train, y_train))
    .shuffle(10000, seed=smp.dp_rank())
    .batch(256, drop_remainder=True)
)

# smdistributed: Define smp.DistributedModel the same way as Keras sub-classing API
class MyModel(smp.DistributedModel):
    def __init__(self):
        super(MyModel, self).__init__()
        # defin layers

    def call(self, x, training=None):
        # define forward pass and return the model output

model = MyModel()

loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
optimizer = tf.keras.optimizers.Adam()
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name="train_accuracy")

# smdistributed: Define smp.step. Return any tensors needed outside
@smpp.step
def get_grads(images, labels):
    predictions = model(images, training=True)
    loss = loss_object(labels, predictions)

```

```

grads = optimizer.get_gradients(loss, model.trainable_variables)
return grads, loss, predictions

@tf.function
def train_step(images, labels):
    gradients, loss, predictions = get_grads(images, labels)

    # smdistributed: Accumulate the gradients across microbatches
    gradients = [g.accumulate() for g in gradients]
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    # smdistributed: Merge predictions and average losses across microbatches
    train_accuracy(labels, predictions.merge())
    return loss.reduce_mean()

for epoch in range(5):
    # Reset the metrics at the start of the next epoch
    train_accuracy.reset_states()
    for images, labels in train_ds:
        loss = train_step(images, labels)
        accuracy = train_accuracy.result()

```

## TensorFlow with Horovod

The SageMaker distributed model parallel library can be used with Horovod for hybrid model and data parallelism. In this case, the total number of GPUs must be divisible by the number of partitions. The quotient is inferred to be the number of model replicas or data parallelism degree. For instance, if a training job is launched with 8 processes (which corresponds to 8 GPUs), and `partitions` is 2, then the library applies 2-way model parallelism and 4-way data parallelism over the 8 GPUs.

To access the data parallel rank and model parallel rank of a process, you can use `smp.dp_rank()` and `smp.mp_rank()`, respectively. To see all MPI primitives the library exposes, see [MPI Basics](#) in the API documentation.

If you are using Horovod, you should not directly call `hvd.init`. Instead, set "horovod" to `True` in the SageMaker Python SDK `modelparallel` parameters, and the library internally initializes Horovod. This is because Horovod must be initialized based on the device assignments of model partitions, and calling `hvd.init()` directly results in problems.

Furthermore, using `hvd.DistributedOptimizer` directly results in poor performance or hangs, since this implicitly places the AllReduce operation inside `smp.step`. The recommended way to use the model parallel library with Horovod is by directly calling `hvd.allreduce` after calling `accumulate()` or `reduce_mean()` on the gradients returned from `smp.step`, as seen in the following example.

The four main changes needed in the script are:

- Adding `hvd.allreduce`
- Broadcasting variables after the first batch, as required by Horovod
- Setting the "horovod" parameter to `True` in the `modelparallel` dict in the Python SDK
- Seeding shuffling and/or sharding operations in the data pipeline with `smp.dp_rank()`.

To learn more about the SageMaker's distributed model parallel library API, refer to the [API documentation](#).

```

import tensorflow as tf
import horovod.tensorflow as hvd

```

```

# smdistributed: Import TF2.x API
import smdistributed.modelparallel.tensorflow as smp

# smdistributed: Initialize
smp.init()

# Download and load MNIST dataset.
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data(
    "MNIST-data-%d" % smp.rank()
)
x_train, x_test = x_train / 255.0, x_test / 255.0

# Add a channels dimension
x_train = x_train[..., tf.newaxis]
x_test = x_test[..., tf.newaxis]

# smdistributed: Seed the shuffle with smp.dp_rank(), and drop_remainder
# in batching to make sure batch size is always divisible by number of microbatches
train_ds = (
    tf.data.Dataset.from_tensor_slices((x_train, y_train))
    .shuffle(10000, seed=smp.dp_rank())
    .batch(256, drop_remainder=True)
)

# smdistributed: Define smp.DistributedModel the same way as Keras sub-classing API
class MyModel(smp.DistributedModel):
    def __init__(self):
        super(MyModel, self).__init__()
        # define layers

    def call(self, x, training=None):
        # define forward pass and return model outputs

model = MyModel()

loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
optimizer = tf.keras.optimizers.Adam()
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name="train_accuracy")

# smdistributed: Define smp.step. Return any tensors needed outside
@smpp.step
def get_grads(images, labels):
    predictions = model(images, training=True)
    loss = loss_object(labels, predictions)

    grads = optimizer.get_gradients(loss, model.trainable_variables)
    return grads, loss, predictions

@tf.function
def train_step(images, labels, first_batch):
    gradients, loss, predictions = get_grads(images, labels)

    # smdistributed: Accumulate the gradients across microbatches
    # Horovod: Allreduce the accumulated gradients
    gradients = [hvd.allreduce(g.accumulate()) for g in gradients]
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    # Horovod: Broadcast the variables after first batch
    if first_batch:
        hvd.broadcast_variables(model.variables, root_rank=0)
        hvd.broadcast_variables(optimizer.variables(), root_rank=0)

    # smdistributed: Merge predictions across microbatches

```

```

    train_accuracy(labels, predictions.merge())
    return loss.reduce_mean()

for epoch in range(5):
    # Reset the metrics at the start of the next epoch
    train_accuracy.reset_states()

    for batch, (images, labels) in enumerate(train_ds):
        loss = train_step(images, labels, tf.constant(batch == 0))

```

## Manual partitioning with TensorFlow

Use `smp.partition` context managers to place operations in specific partition. Any operation not placed in any `smp.partition` contexts is placed in the `default_partition`. To learn more about the SageMaker's distributed model parallel library API, refer to the [API documentation](#).

```

import tensorflow as tf

# smdistributed: Import TF2.x API.
import smdistributed.modelparallel.tensorflow as smp

# smdistributed: Initialize
smp.init()

# Download and load MNIST dataset.
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data(
    "MNIST-data-%d" % smp.rank()
)
x_train, x_test = x_train / 255.0, x_test / 255.0

# Add a channels dimension
x_train = x_train[..., tf.newaxis]
x_test = x_test[..., tf.newaxis]

# smdistributed: If needed, seed the shuffle with smp.dp_rank(), and drop_remainder
# in batching to make sure batch size is always divisible by number of microbatches.
train_ds = (
    tf.data.Dataset.from_tensor_slices((x_train, y_train))
    .shuffle(10000, seed=smp.dp_rank())
    .batch(256, drop_remainder=True)
)

# smdistributed: Define smp.DistributedModel the same way as Keras sub-classing API.
class MyModel(smp.DistributedModel):
    def __init__(self):
        # define layers

    def call(self, x):
        with smp.partition(0):
            x = self.layer0(x)
        with smp.partition(1):
            return self.layer1(x)

model = MyModel()

loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
optimizer = tf.keras.optimizers.Adam()
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name="train_accuracy")

# smdistributed: Define smp.step. Return any tensors needed outside
@smmp.step
def get_grads(images, labels):

```

```
predictions = model(images, training=True)
loss = loss_object(labels, predictions)

grads = optimizer.get_gradients(loss, model.trainable_variables)
return grads, loss, predictions

@tf.function
def train_step(images, labels):
    gradients, loss, predictions = get_grads(images, labels)

    # smdistributed: Accumulate the gradients across microbatches
    gradients = [g.accumulate() for g in gradients]
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))

    # smdistributed: Merge predictions and average losses across microbatches
    train_accuracy(labels, predictions.merge())
    return loss.reduce_mean()

for epoch in range(5):
    # Reset the metrics at the start of the next epoch
    train_accuracy.reset_states()
    for images, labels in train_ds:
        loss = train_step(images, labels)
    accuracy = train_accuracy.result()
```

## Modify a PyTorch Training Script

The following are examples of training scripts that you can use to configure SageMaker's model parallel library with PyTorch versions 1.7.1 and 1.6.0, with auto-partitioning and manual partitioning. We recommend that you review the [Important Considerations \(p. 1810\)](#) and [Unsupported Framework Features \(p. 1812\)](#) before creating a training script.

The required modifications you must make to your training script to use the library are listed in [PyTorch \(p. 1813\)](#).

If you want to use manual partitioning, also review [Manual Partitioning with PyTorch \(p. 1814\)](#).

### Tip

For end to end, runnable notebook examples that demonstrate how to use a TensorFlow training script with the SageMaker distributed model parallel library, see [PyTorch Examples \(p. 1819\)](#).

Note that auto-partitioning is enabled by default. Unless otherwise specified, the following scripts use auto-partitioning.

### Topics

- [Important Considerations \(p. 1810\)](#)
- [Unsupported Framework Features \(p. 1812\)](#)
- [PyTorch \(p. 1813\)](#)
- [Manual Partitioning with PyTorch \(p. 1814\)](#)

### Important Considerations

When you configure a PyTorch training script using SageMaker's distributed model parallel library, you should be aware of the following:

- If you are using an optimization technique that relies on global gradient norms, for example gradient norm from the entire model, such as some variants of LAMB optimizer or global gradient clipping,

you need to gather all the norms across the model partitions for correctness. You can use the library's communication basic data types to do this.

- All `torch.Tensor` arguments to the forward methods of the `nn.Modules` in your model must be used in the computation of the module output. In other words, the library does not support the case where there is a `torch.Tensor` argument to a module on which the module output does not depend.
- The argument to the `smp.DistributedModel.backward()` call must depend on all model outputs. In other words, there cannot be an output from the `smp.DistributedModel.forward` call that is not used in the computation of the tensor that is fed into the `smp.DistributedModel.backward` call.
- If there are `torch.cuda.synchronize()` calls in your code, you might need to call `torch.cuda.set_device(smp.local_rank())` immediately before the `synchronize` call. Otherwise unnecessary CUDA contexts might be created in device 0, which will needlessly consume memory.
- Since the library places `nn.Modules` on different devices, the modules in the model must not depend on any global state that is modified inside `smp.step`. Any state that remains fixed throughout training, or that is modified outside `smp.step` in a way that is visible to all processes, is allowed.
- You don't need to move the model to GPU (for example, using `model.to(device)`) when using the library. If you try to move the model to GPU before the model is partitioned (before the first `smp.step` call), the move call is ignored. The library automatically moves the part of the model assigned to a rank to its GPU. Once training with the library starts, don't move the model to CPU and use it, as it won't have correct parameters for modules not assigned to the partition held by the process. If you want to retrain a model or use it for inference without the library after it was trained using the model parallel library, the recommended way is to save the full model using our checkpointing API and load it back to a regular PyTorch Module.
- If you have a list of modules such that output of one feeds into another, replacing that list with `nn.Sequential` can significantly improve performance.
- The weight update (`optimizer.step()`) needs to happen outside of `smp.step` because that is when the entire backward pass is done and gradients are ready. When using a hybrid model with model and data parallelism, at this point, Allreduce of gradients is also guaranteed to finish.
- When using the library in combination with data parallelism, make sure that the number of batches on all data parallel ranks is the same so that Allreduce does not hang waiting for a rank which is not participating in the step.
- If you launch a training job using an `ml.p4d` instance type (such as `ml.p4d.24xlarge`), you must set the data loader variable `num_workers=0`. For example, you may define your `DataLoader` as follows:

```
dataloader = torch.utils.data.DataLoader(
    data,
    batch_size=batch_size,
    num_workers=0,
    pin_memory=True,
    drop_last=True,
    shuffle=shuffle,
)
```

- The inputs to `smp.step` must be the model inputs generated by `DataLoader`. This is because `smp.step` internally splits the input tensors along the batch dimension and pipelines them. This means that passing `DataLoader` itself to the `smp.step` function to generate the model inputs inside does not work.

For example, if you define a `DataLoader` as follows:

```
train_loader = torch.utils.data.DataLoader(dataset, batch_size=64, drop_last=True)
```

You should access the model inputs generated by `train_loader` and pass those to an `smp.step` decorated function. Do not pass `train_loader` directly to the `smp.step` function.

```

def train(model, device, train_loader, optimizer):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        ...
        _, loss_mb = train_step(model, data, target)
        ...

@smp.step
def train_step(model, data, target):
    ...
    return output, loss

```

- The input tensors to `smp.step` must be moved to the current device using `.to()` API, which must take place after the `torch.cuda.set_device(local_rank())` call.

For example, you may define the `train` function as follows. This function adds `data` and `target` to the current device using `.to()` API before using those input tensors to call `train_step`.

```

def train(model, device, train_loader, optimizer):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        # smdistributed: Move input tensors to the GPU ID used by the current process,
        # based on the set_device call.
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        # Return value, loss_mb is a StepOutput object
        _, loss_mb = train_step(model, data, target)

        # smdistributed: Average the loss across microbatches.
        loss = loss_mb.reduce_mean()

    optimizer.step()

```

The input tensors to this `smp.set` decorated function have been moved to the current device in the `train` function above. The model does *not* need to be moved to the current device. The library automatically moves the part of the model assigned to a rank to its GPU.

```

@smp.step
def train_step(model, data, target):
    output = model(data)
    loss = F.nll_loss(output, target, reduction="mean")
    model.backward(loss)
    return output, loss

```

## Unsupported Framework Features

The following PyTorch features are unsupported by SageMaker's distributed model parallel library:

- When using data parallelism with DDP, the `DistributedDataParallel` wrapper is not supported. The library internally manages integrating with DDP, including parameter broadcast and gradient Allreduce. When using the library, module buffers are only broadcast once at the start of training. If your model has module buffers and you require module buffers to be synchronized across data parallel groups at each step, you can do so through the `torch.distributed` API, using the process group that can be obtained via `smp.get_dp_process_group()`.
- For mixed precision training, the `apex.amp` module is not supported. The recommended way to use the library with automatic mixed-precision is to use `torch.cuda.amp`, with the exception of using `smp.amp.GradScaler` instead of the implementation in torch.
- `torch.jit.ScriptModules` or `ScriptFunctions` are not supported by `smp.DistributedModel`.

- apex : FusedLayerNorm, FusedAdam, FusedLAMB, and FusedNovoGrad from apex are not supported. You can use the library implementations of these through smp.optimizers and smp.nn APIs instead.

## PyTorch

The following training script changes are required to run a PyTorch model with SageMaker's distributed model parallel library:

1. Import and initialize the library with `smp.init()`.
2. Wrap the model with `smp.DistributedModel`. Be mindful that any tensors returned from the forward method of the underlying `nn.Module` object will be broadcast across model-parallel devices, incurring communication overhead, so any tensors that are not needed outside the call method (such as intermediate activations) should not be returned.
3. Wrap the optimizer with `smp.DistributedOptimizer`.
4. Use the returned `DistributedModel` object instead of a user model.
5. Put the forward and backward logic in a step function and decorate it with `smp.step`.
6. Restrict each process to its own device through `torch.cuda.set_device(smp.local_rank())`.
7. Move the input tensors to the GPU using the `.to()` API before the `smp.step` call (see example below).
8. Replace `torch.Tensor.backward` and `torch.autograd.backward` with `DistributedModel.backward`.
9. Perform post-processing on the outputs across microbatches using `StepOutput` methods such as `reduce_mean`.
- 10 If there is an evaluation step, similarly place the forward logic inside an `smp.step`-decorated function and post-process the outputs using `StepOutput API`.
- 11 Set `drop_last=True` in `DataLoader`. Alternatively, manually skip a batch in the training loop if the batch size is not divisible by the number of microbatches.

To learn more about the SageMaker's distributed model parallel library API, refer to the [API documentation](#).

```

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchnet.dataset import SplitDataset
from torchvision import datasets

import smdistributed.modelparallel.torch as smp

class GroupedNet(nn.Module):
    def __init__(self):
        super(GroupedNet, self).__init__()
        # define layers

    def forward(self, x):
        # define forward pass and return model outputs

# smdistributed: Define smp.step. Return any tensors needed outside.
@smp.step
def train_step(model, data, target):
    output = model(data)
    loss = F.nll_loss(output, target, reduction="mean")
    model.backward(loss)

```

```

        return output, loss

def train(model, device, train_loader, optimizer):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        # smdistributed: Move input tensors to the GPU ID used by the current process,
        # based on the set_device call.
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        # Return value, loss_mb is a StepOutput object
        _, loss_mb = train_step(model, data, target)

        # smdistributed: Average the loss across microbatches.
        loss = loss_mb.reduce_mean()

        optimizer.step()

# smdistributed: initialize the backend
smp.init()

# smdistributed: Set the device to the GPU ID used by the current process.
# Input tensors should be transferred to this device.
torch.cuda.set_device(smp.local_rank())
device = torch.device("cuda")

# smdistributed: Download only on a single process per instance.
# When this is not present, the file is corrupted by multiple processes trying
# to download and extract at the same time
dataset = datasets.MNIST("../data", train=True, download=False)

# smdistributed: Shard the dataset based on data-parallel ranks
if smp.dp_size() > 1:
    partitions_dict = {f"{i}": 1 / smp.dp_size() for i in range(smp.dp_size())}
    dataset = SplitDataset(dataset, partitions=partitions_dict)
    dataset.select(f"{{smp.dp_rank()}}")

# smdistributed: Set drop_last=True to ensure that batch size is always divisible
# by the number of microbatches
train_loader = torch.utils.data.DataLoader(dataset, batch_size=64, drop_last=True)

model = GroupedNet()
optimizer = optim.Adadelta(model.parameters(), lr=4.0)

# smdistributed: Use the DistributedModel container to provide the model
# to be partitioned across different ranks. For the rest of the script,
# the returned DistributedModel object should be used in place of
# the model provided for DistributedModel class instantiation.
model = smp.DistributedModel(model)
optimizer = smp.DistributedOptimizer(optimizer)

train(model, device, train_loader, optimizer)

```

## Manual Partitioning with PyTorch

Use `smp.partition` context managers to place modules in specific devices. Any module not placed in any `smp.partition` contexts is placed in the `default_partition`. The `default_partition` needs to be provided if `auto_partition` is set to `False`. The modules that are created within a specific `smp.partition` context are placed on the corresponding partition.

To learn more about the SageMaker's distributed model parallel library API, refer to the [API documentation](#).

```
import torch
```

```

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchnet.dataset import SplitDataset
from torchvision import datasets

import smdistributed.modelparallel.torch as smp

class GroupedNet(nn.Module):
    def __init__(self):
        super(GroupedNet, self).__init__()
        with smp.partition(0):
            # define child modules on device 0
        with smp.partition(1):
            # define child modules on device 1

    def forward(self, x):
        # define forward pass and return model outputs

# smdistributed: Define smp.step. Return any tensors needed outside.
@smp.step
def train_step(model, data, target):
    output = model(data)
    loss = F.nll_loss(output, target, reduction="mean")
    model.backward(loss)
    return output, loss

def train(model, device, train_loader, optimizer):
    model.train()
    for batch_idx, (data, target) in enumerate(train_loader):
        # smdistributed: Move input tensors to the GPU ID used by the current process,
        # based on the set_device call.
        data, target = data.to(device), target.to(device)
        optimizer.zero_grad()
        # Return value, loss_mb is a StepOutput object
        _, loss_mb = train_step(model, data, target)

        # smdistributed: Average the loss across microbatches.
        loss = loss_mb.reduce_mean()

        optimizer.step()

# smdistributed: initialize the backend
smp.init()

# smdistributed: Set the device to the GPU ID used by the current process.
# Input tensors should be transferred to this device.
torch.cuda.set_device(smp.local_rank())
device = torch.device("cuda")

# smdistributed: Download only on a single process per instance.
# When this is not present, the file is corrupted by multiple processes trying
# to download and extract at the same time
dataset = datasets.MNIST("../data", train=True, download=False)

# smdistributed: Shard the dataset based on data-parallel ranks
if smp.dp_size() > 1:
    partitions_dict = {f"{i}": 1 / smp.dp_size() for i in range(smp.dp_size())}
    dataset = SplitDataset(dataset, partitions=partitions_dict)
    dataset.select(f"{smp.dp_rank()}")

# smdistributed: Set drop_last=True to ensure that batch size is always divisible
# by the number of microbatches
train_loader = torch.utils.data.DataLoader(dataset, batch_size=64, drop_last=True)

```

```
model = GroupedNet()
optimizer = optim.Adadelta(model.parameters(), lr=4.0)

# smdistributed: Use the DistributedModel container to provide the model
# to be partitioned across different ranks. For the rest of the script,
# the returned DistributedModel object should be used in place of
# the model provided for DistributedModel class instantiation.
model = smp.DistributedModel(model)
optimizer = smp.DistributedOptimizer(optimizer)

train(model, device, train_loader, optimizer)
```

## SageMaker distributed model parallel Configuration Tips and Pitfalls

Review the following tips and pitfalls before using Amazon SageMaker's distributed model parallel library. This list includes tips that are applicable across frameworks. For TensorFlow and PyTorch specific tips, see [Modify a TensorFlow Training Script \(p. 1805\)](#) and [Modify a PyTorch Training Script \(p. 1810\)](#), respectively.

### Batch Size and Number of Microbatches

- The library is most efficient when the batch size is increased. For use cases where the model fits within a single device, but can only be trained with a small batch size, batch size can and should be increased after the library is integrated. Model parallelism saves memory for large models, enabling you to train using batch sizes that previously did not fit in memory.
- Choosing a number of microbatches that is too small or too large can hurt performance. The library executes each microbatch sequentially in each device, so microbatch size (batch size divided by number of microbatches) must be large enough to fully utilize each GPU. At the same time, pipeline efficiency increases with the number of microbatches, so striking the right balance is important. Typically, a good starting point is to try 2 or 4 microbatches, increasing the batch size to the memory limit, and then experiment with larger batch sizes and numbers of microbatches. As the number of microbatches is increased, larger batch sizes might become feasible if an interleaved pipeline is used.
- Your batch size must be always divisible by the number of microbatches. Note that depending on the size of the dataset, sometimes the last batch of every epoch can be of a smaller size than the rest, and this smaller batch needs to be divisible by the number of microbatches as well. If it is not, you can set `drop_remainder=True` in the `tf.Dataset.batch()` call (in TensorFlow), or set `drop_last=True` in `DataLoader` (in PyTorch), so that this last, small batch is not used. If you are using a different API for the data pipeline, you might need to manually skip the last batch whenever it is not divisible by the number of microbatches.

### Manual Partitioning

- If you use manual partitioning, be mindful of the parameters that are consumed by multiple operations and modules in your model, such as the embedding table in transformer architectures. Modules that share the same parameter must be placed in the same device for correctness. When auto-partitioning is used, the library automatically enforces this constraint.

### Data Preparation

- If the model takes multiple inputs, make sure you seed the random operations in your data pipeline (e.g., shuffling) with `smp.dp_rank()`. If the dataset is being deterministically sharded across data parallel devices, make sure that the shard is indexed by `smp.dp_rank()`. This is to make sure that the order of the data seen on all ranks that form a model partition is consistent.

### Returning Tensors from `smp.DistributedModel`

- Any tensor that is returned from the `smp.DistributedModel.call` (for TensorFlow) or `smp.DistributedModel.forward` (for PyTorch) function is broadcast to all other ranks, from the rank that computed that particular tensor. As a result, any tensor that is not needed outside the call and forward methods (intermediate activations, for example) should not be returned, as this causes needless communication and memory overhead and hurts performance.

### The `@smp.step` Decorator

- If an `smp.step`-decorated function has a tensor argument that does not have a batch dimension, the argument name must be provided in the `non_split_inputs` list when calling `smp.step`. This prevents the library from attempting to split the tensor into microbatches. For more information see `smp.step` in the API documentation.

## Model Parallel Troubleshooting

If you run into an error, you can use the following list to try to troubleshoot your training job. If the problem persists, contact Amazon Support.

### Topics

- [Considerations for Using SageMaker Debugger with SageMaker Distributed Model Parallel \(p. 1817\)](#)
- [Saving Checkpoints \(p. 1818\)](#)
- [Convergence Using Model Parallel and TensorFlow \(p. 1819\)](#)

## Considerations for Using SageMaker Debugger with SageMaker Distributed Model Parallel

SageMaker Debugger is not available for SageMaker distributed model parallel. Debugger is enabled by default for all SageMaker TensorFlow and PyTorch training jobs, and you might see an error that looks like the following:

```
FileNotFoundException: [Errno 2] No such file or directory: '/opt/ml/checkpoints/metadata.json.sagemaker-uploading'
```

To fix this issue, disable Debugger by passing `debugger_hook_config=False` when creating a framework estimator as shown in the following example.

```
bucket=sagemaker.Session().default_bucket()
base_job_name="sagemaker-checkpoint-test"
checkpoint_in_bucket="checkpoints"

# The S3 URI to store the checkpoints
checkpoint_s3_bucket="s3://{}//{}//{}".format(bucket, base_job_name, checkpoint_in_bucket)

estimator = TensorFlow(
    ...
    distribution={"smdistributed": {"modelparallel": { "enabled": True }}},
    checkpoint_s3_uri=checkpoint_s3_bucket,
    checkpoint_local_path="/opt/ml/checkpoints",
    debugger_hook_config=False
)
```

## Saving Checkpoints

You might run into the following error when saving checkpoints of a large model on SageMaker:

```
InternalServerError: We encountered an internal error. Please try again
```

This could be caused by a SageMaker limitation while uploading the local checkpoint to Amazon S3 during training. To disable checkpointing in SageMaker, use the following example to explicitly upload the checkpoints.

If you run into the preceding error, do not use `checkpoint_s3_uri` with the SageMaker estimator call. While saving checkpoints for larger models, we recommend saving checkpoints to a custom directory and passing the same to the helper function (as a `local_path` argument).

```
import os

def aws_s3_sync(source, destination):
    """aws s3 sync in quiet mode and time profile"""
    import time, subprocess
    cmd = ["aws", "s3", "sync", "--quiet", source, destination]
    print(f"Syncing files from {source} to {destination}")
    start_time = time.time()
    p = subprocess.Popen(cmd, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    p.wait()
    end_time = time.time()
    print("Time Taken to Sync: ", (end_time-start_time))
    return

def sync_local_checkpoints_to_s3(local_path="/opt/ml/checkpoints",
                                 s3_uri=os.path.dirname(os.path.dirname(os.getenv('SM_MODULE_DIR', '')))+'/checkpoints'):
    """ sample function to sync checkpoints from local path to s3 """

    import boto3
    #check if local path exists
    if not os.path.exists(local_path):
        raise RuntimeError("Provided local path {local_path} does not exist. Please check")

    #check if s3 bucket exists
    s3 = boto3.resource('s3')
    if not s3_uri.startswith("s3://"):
        raise ValueError(f"Provided s3 uri {s3_uri} is not valid.")

    s3_bucket = s3_uri.replace('s3://','').split('/')[0]
    print(f"S3 Bucket: {s3_bucket}")
    try:
        s3.meta.client.head_bucket(Bucket=s3_bucket)
    except Exception as e:
        raise e
    aws_s3_sync(local_path, s3_uri)
    return

def sync_s3_checkpoints_to_local(local_path="/opt/ml/checkpoints",
                                 s3_uri=os.path.dirname(os.path.dirname(os.getenv('SM_MODULE_DIR', '')))+'/checkpoints'):
    """ sample function to sync checkpoints from s3 to local path """

    import boto3
    #try to create local path if it does not exist
    if not os.path.exists(local_path):
        print(f"Provided local path {local_path} does not exist. Creating...")
        try:
            os.makedirs(local_path)
        except Exception as e:
            raise RuntimeError(f"Failed to create {local_path}")
```

```
#check if s3 bucket exists
s3 = boto3.resource('s3')
if not s3_uri.startswith("s3://"):
    raise ValueError(f"Provided s3 uri {s3_uri} is not valid.")

s3_bucket = s3_uri.replace('s3://','').split('/')[0]
print(f"S3 Bucket: {s3_bucket}")
try:
    s3.meta.client.head_bucket(Bucket=s3_bucket)
except Exception as e:
    raise e
aws_s3_sync(s3_uri, local_path)
return
```

Usage of helper functions:

```
#base_s3_uri - user input s3 uri or save to model directory (default)
#curr_host - to save checkpoints of current host
#iteration - current step/epoch during which checkpoint is saved

# save checkpoints on every node using local_rank
if smp.local_rank() == 0:
    base_s3_uri = os.path.dirname(os.path.dirname(os.getenv('SM_MODULE_DIR', '')))
    curr_host = os.environ['SM_CURRENT_HOST']
    full_s3_uri = f'{base_s3_uri}/checkpoints/{curr_host}/{iteration}'
    sync_local_checkpoints_to_s3(local_path=checkpoint_dir, s3_uri=full_s3_uri)
```

## Convergence Using Model Parallel and TensorFlow

When you use SageMaker multi-node training with TensorFlow and distributed model parallel, the loss may not converge as expected because the order of training input files may be different on each node. This may cause different ranks in the same model parallel group to work on different input files, causing inconsistencies. To prevent this, ensure the input files are ordered the same way in all the ranks before they get converted to TensorFlow datasets. One way to achieve this is to sort the input file names in the training script.

## Distributed Training Jupyter Notebook Examples

The following notebooks provide example Amazon SageMaker distributed implementations for popular deep learning frameworks and models. For vision (image) models, try MNIST or MaskRCNN. For language (text) models, try BERT.

These notebooks are provided in the [SageMaker examples repository](#). You can also browse them on the [SageMaker examples website](#).

The examples are set up to use p3.16xlarge instances for the worker nodes, but you may choose other p3 instance types if you wish. You can test a notebook using a cluster with only 1 node, but to see any performance benefit, you should use a cluster of multiple nodes (2 or more). The examples call out the section in which you modify this configuration.

## PyTorch Examples

### SageMaker Distributed Data Parallel

- MNIST with PyTorch 1.6 and SageMaker Data Parallel
- MaskRCNN with PyTorch 1.6 and SageMaker Data Parallel
- BERT with PyTorch 1.6 and SageMaker Data Parallel

### SageMaker Distributed Model Parallel

- MNIST with PyTorch 1.6 and SageMaker Model Parallel
- BERT with PyTorch 1.6 and SageMaker Model Parallel

## TensorFlow Examples

### SageMaker Distributed Data Parallel

- MNIST with TensorFlow 2.3.1 and SageMaker Data Parallel
- MaskRCNN with TensorFlow 2.3.1 and SageMaker Data Parallel
- BERT with TensorFlow 2.3.1 and SageMaker Data Parallel

### SageMaker Distributed Model Parallel

- MNIST with TensorFlow 2.3.1 and SageMaker Model Parallel

## Use a SageMaker Notebook Instance

To use the provided examples, we recommend that you use an Amazon SageMaker notebook instance. A notebook instance is a machine learning (ML)-optimized Amazon EC2 instance running the Jupyter Notebook and JupyterServer apps. If you do not have an active notebook instance, follow the instructions in [Create a Notebook Instance \(p. 120\)](#) in the SageMaker developer guide to create one.

After you have created an instance, in the **Notebook instances** area of the SageMaker console, do the following:

1. Open **JupyterLab**.
2. Select the examples icon (  ) in the left tray.
3. Browse the examples for **Training** and look for notebooks titled *Distributed Data Parallel* or *Distributed Model Parallel*.

## Use SageMaker Studio

You can run these example Jupyter Notebooks in SageMaker Studio. To download and use an example notebook, do the following in Studio:

1. Open a terminal.
2. In the command line, navigate to the SageMaker folder.

```
$ cd SageMaker
```

3. Clone the SageMaker examples repo.

```
git clone https://github.com/aws/amazon-sagemaker-examples.git
```

4. In the JupyterLab interface, navigate into the `amazon-sagemaker-examples` folder.
5. In the `training/distributed_training` folder, there are folders for frameworks, and in each of these, there are folders for `data_parallel` and `model_parallel`. Choose the example of your choice and follow the instructions to launch distributed training with an SageMaker distributed training library.

# Detect Posttraining Data and Model Bias

Posttraining bias analysis can help reveal biases that might have emanated from biases in the data, or from biases introduced by the classification and prediction algorithms. These analyses take into consideration the data, including the labels, and the predictions of a model. You assess performance by analyzing predicted labels or by comparing the predictions with the observed target values in the data with respect to groups with different attributes. There are different notions of fairness, each requiring different bias metrics to measure.

There are legal concepts of fairness that might not be easy to capture because they are hard to detect. For example, the US concept of disparate impact that occurs when a group, referred to as a less favored facet  $d$ , experiences an adverse effect even when the approach taken appears to be fair. This type of bias might not be due to a machine learning model, but might still be detectable by posttraining bias analysis.

Amazon SageMaker Clarify tries to ensure a consistent use of terminology. For a list of terms and their definitions, see [Amazon SageMaker Clarify Terms for Bias and Fairness \(p. 561\)](#).

For additional information about posttraining bias metrics, see [Fairness Measures for Machine Learning in Finance](#).

## Sample Notebooks

Amazon SageMaker Clarify provides the following sample notebook for posttraining bias detection:

- [Explainability and bias detection with Amazon SageMaker Clarify](#) – Use SageMaker Clarify to create a processing job for the detecting bias and explaining model predictions with feature attributions.

This notebook has been verified to run in Amazon SageMaker Studio only. If you need instructions on how to open a notebook in Amazon SageMaker Studio, see [Create or Open an Amazon SageMaker Studio Notebook \(p. 79\)](#). If you're prompted to choose a kernel, choose **Python 3 (Data Science)**.

### Topics

- [Measure Posttraining Data and Model Bias \(p. 1821\)](#)
- [Configure an Amazon SageMaker Clarify Processing Jobs for Fairness and Explainability \(p. 1840\)](#)
- [Run SageMaker Clarify Processing Jobs for Bias Analysis and Explainability \(p. 1847\)](#)
- [Troubleshoot SageMaker Clarify Processing Jobs \(p. 1850\)](#)

## Measure Posttraining Data and Model Bias

Amazon SageMaker Clarify provides eleven posttraining data and model bias metrics to help quantify various conceptions of fairness. These concepts cannot all be satisfied simultaneously and the selection depends on specifics of the cases involving potential bias being analyzed. Most of these metrics are a combination of the numbers taken from the binary classification confusion matrices for the different demographic groups. Because fairness and bias can be defined by a wide range of metrics, human judgment is required to understand and choose which metrics are relevant to the individual use case, and customers should consult with appropriate stakeholders to determine the appropriate measure of fairness for their application.

We use the following notation to discuss the bias metrics. The conceptual model described here is for binary classification, where events are labeled as having only two possible outcomes in their sample space, referred to as positive (with value 1) and negative (with value 0). This framework is usually extensible to multiclass classification in a straightforward way or to cases involving continuous valued outcomes when needed. In the binary classification case, positive and negative labels are assigned

to outcomes recorded in a raw dataset for a favored facet  $a$  and for a disfavored facet  $d$ . These labels  $y$  are referred to as *observed labels* to distinguish them from the *predicted labels*  $y'$  that are assigned by a machine learning model during the training or inferences stages of the ML lifecycle. These labels are used to define probability distributions  $P_a(y)$  and  $P_d(y)$  for their respective facet outcomes.

- **labels:**
  - $y$  represents the  $n$  observed labels for event outcomes in a training dataset.
  - $y'$  represents the predicted labels for the  $n$  observed labels in the dataset by a trained model.
- **outcomes:**
  - A positive outcome (with value 1) for a sample, such as an application acceptance.
    - $n^{(1)}$  is the number of observed labels for positive outcomes (acceptances).
    - $n'^{(1)}$  is the number of predicted labels for positive outcomes (acceptances).
  - A negative outcome (with value 0) for a sample, such as an application rejection.
    - $n^{(0)}$  is the number of observed labels for negative outcomes (rejections).
    - $n'^{(0)}$  is the number of predicted labels for negative outcomes (rejections).
- **facet values:**
  - facet  $a$  – The feature value that defines a demographic that bias favors.
    - $n_a$  is the number of observed labels for the favored facet value:  $n_a = n_a^{(1)} + n_a^{(0)}$  the sum of the positive and negative observed labels for the value facet  $a$ .
    - $n'_a$  is the number of predicted labels for the favored facet value:  $n'_a = n'_a^{(1)} + n'_a^{(0)}$  the sum of the positive and negative predicted outcome labels for the facet value  $a$ . Note that  $n'_a = n_a$ .
  - facet  $d$  – The feature value that defines a demographic that bias disfavors.
    - $n_d$  is the number of observed labels for the disfavored facet value:  $n_d = n_d^{(1)} + n_d^{(0)}$  the sum of the positive and negative observed labels for the facet value  $d$ .
    - $n'_d$  is the number of predicted labels for the disfavored facet value:  $n'_d = n'_d^{(1)} + n'_d^{(0)}$  the sum of the positive and negative predicted labels for the facet value  $d$ . Note that  $n'_d = n_d$ .
- **probability distributions for outcomes of the labeled facet data outcomes:**
  - $P_a(y)$  is the probability distribution of the observed labels for facet  $a$ . For binary labeled data, this distribution is given by the ratio of the number of samples in facet  $a$  labeled with positive outcomes to the total number,  $P_a(y^1) = n_a^{(1)} / n_a$ , and the ratio of the number of samples with negative outcomes to the total number,  $P_a(y^0) = n_a^{(0)} / n_a$ .
  - $P_d(y)$  is the probability distribution of the observed labels for facet  $d$ . For binary labeled data, this distribution is given by the number of samples in facet  $d$  labeled with positive outcomes to the total number,  $P_d(y^1) = n_d^{(1)} / n_d$ , and the ratio of the number of samples with negative outcomes to the total number,  $P_d(y^0) = n_d^{(0)} / n_d$ .

The following table contains a cheat sheet for quick guidance and links to the posttraining bias metrics.

### Posttraining Bias Metrics

posttraining bias metric	Description	Example question	Interpreting metric values
Difference in Positive Proportions in Predicted Labels (DPPL) (p. 1830)	Measures the difference in the proportion of positive predictions between the favored facet $a$ and the disfavored facet $d$ .	Has there been an imbalance across demographic groups in the predicted positive outcomes that might indicate bias?	Range for normalized binary & multiclass facet labels: $[-1, +1]$ Range for continuous labels: $(-\infty, +\infty)$ Interpretation:

posttraining bias metric	Description	Example question	Interpreting metric values
			<ul style="list-style-type: none"> <li>Positive values indicate that the favored facet <math>a</math> has a higher proportion of predicted positive outcomes.</li> <li>Values near zero indicate a more equal proportion of predicted positive outcomes between facets.</li> <li>Negative values indicate the disfavored facet <math>d</math> has a higher proportion of predicted positive outcomes.</li> </ul>
<a href="#">Disparate Impact (DI) (p. 1831)</a>	Measures the ratio of proportions of the predicted labels for the favored facet $a$ and the disfavored facet $d$ .	Has there been an imbalance across demographic groups in the predicted positive outcomes that might indicate bias?	<p>Range for normalized binary, multiclass, and continuous labels: <math>[0, \infty)</math></p> <p>Interpretation:</p> <ul style="list-style-type: none"> <li>Values greater than 1 indicate the favored facet <math>a</math> has a higher proportion of predicted positive outcomes.</li> <li>A value of 1 indicates that we have demographic parity.</li> <li>Values less than 1 indicate the disfavored facet <math>d</math> has a higher proportion of predicted positive outcomes.</li> </ul>

posttraining bias metric	Description	Example question	Interpreting metric values
<a href="#">Difference in Conditional Acceptance (DCAcc) (p. 1831)</a>	Compares the observed labels to the labels predicted by a model and assesses whether this is the same across facets for predicted positive outcomes (acceptances).	Are there more or less acceptances for loan applications than predicted for one age group as compared to another based on qualifications?	<p>The range for binary, multicategory facet, and continuous labels: <math>(-\infty, +\infty)</math>.</p> <ul style="list-style-type: none"> <li>Positive values indicate a possible bias against the qualified applicants from the disfavored facet <math>d</math>.</li> <li>Values near zero indicate that qualified applicants from both facets are being accepted in a similar way.</li> <li>Negative values indicate a possible bias against the qualified applicants from the favored facet <math>a</math>.</li> </ul>
<a href="#">Difference in Conditional Rejection (DCR) (p. 1832)</a>	Compares the observed labels to the labels predicted by a model and assesses whether this is the same across facets for negative outcomes (rejections).	Are there more or less rejections for loan applications than predicted for one age group as compared to another based on qualifications?	<p>The range for binary, multicategory facet, and continuous labels: <math>(-\infty, +\infty)</math>.</p> <ul style="list-style-type: none"> <li>Positive values indicate a possible bias against the qualified applicants from the disfavored facet <math>d</math>.</li> <li>Values near zero indicate that qualified applicants from both facets are being rejected in a similar way.</li> <li>Negative values indicate a possible bias against the qualified applicants from the favored facet <math>a</math>.</li> </ul>

posttraining bias metric	Description	Example question	Interpreting metric values
<a href="#">Recall Difference (RD) (p. 1834)</a>	Compares the recall of the model for the favored and disfavored facets.	Is there an age-based bias in lending due to a model having higher recall for one age group as compared to another?	<p>Range for binary and multiclass classification: [-1, +1].</p> <ul style="list-style-type: none"> <li>Positive values suggest that the model finds more of the true positives for facet <math>a</math> and is biased against the disfavored facet <math>d</math>.</li> <li>Values near zero suggest that the model finds about the same number of true positives in both facets and is not biased.</li> <li>Negative values suggest that the model finds more of the true positives for facet <math>d</math> and is biased against the favored facet <math>a</math>.</li> </ul>

posttraining bias metric	Description	Example question	Interpreting metric values
<a href="#">Difference in Acceptance Rates (DAR) (p. 1835)</a>	Measures the difference in the ratios of the observed positive outcomes (TP) to the predicted positives (TP + FP) between the favored and disfavored facets.	Does the model have equal precision when predicting loan acceptances for qualified applicants across all age groups?	<p>The range for binary, multicategory facet, and continuous labels is [-1, +1].</p> <ul style="list-style-type: none"> <li>Positive values indicate a possible bias against facet <math>d</math> caused by the occurrence of relatively more false positives in the disfavored facet <math>d</math>.</li> <li>Values near zero indicate the observed labels for positive outcomes (acceptances) are being predicted with equal precision for both facets by the model.</li> <li>Negative values indicate a possible bias against facet <math>a</math> caused by the occurrence of relatively more false positives in the favored facet <math>a</math>.</li> </ul>

<b>posttraining bias metric</b>	<b>Description</b>	<b>Example question</b>	<b>Interpreting metric values</b>
<a href="#">Difference in Rejection Rates (DRR) (p. 1835)</a>	Measures the difference in the ratios of the observed negative outcomes ( $TN$ ) to the predicted negatives ( $TN + FN$ ) between the disfavored and favored facets.	Does the model have equal precision when predicting loan rejections for unqualified applicants across all age groups?	<p>The range for binary, multicategory facet, and continuous labels is <math>[-1, +1]</math>.</p> <ul style="list-style-type: none"> <li>Positive values indicate a possible bias caused by the occurrence of relatively more false negatives in the favored facet <math>a</math>.</li> <li>Values near zero indicate the observed labels for negative outcomes (rejections) are being predicted with equal precision for both facets by the model.</li> <li>Negative values indicate a possible bias caused by the occurrence of relatively more false negatives in the disfavored facet <math>d</math>.</li> </ul>

posttraining bias metric	Description	Example question	Interpreting metric values
<a href="#">Accuracy Difference (AD) (p. 1836)</a>	Measures the difference between the prediction accuracy for the favored and disfavored facets.	Does the model predict labels as accurately for applications across all demographic groups?	<p>The range for binary and multicategory facet labels is <math>[-1, +1]</math>.</p> <ul style="list-style-type: none"> <li>Positive values indicate that facet <math>d</math> suffers more from some combination of false positives (Type I errors) or false negatives (Type II errors). This means there is a potential bias against the disfavored facet <math>d</math>.</li> <li>Values near zero occur when the prediction accuracy for facet <math>a</math> is similar to that for facet <math>d</math>.</li> <li>Negative values indicate that facet <math>a</math> suffers more from some combination of false positives (Type I errors) or false negatives (Type II errors). This means there is a bias against the favored facet <math>a</math>.</li> </ul>
<a href="#">Treatment Equality (TE) (p. 1837)</a>	Measures the difference in the ratio of false positives to false negatives between the favored and disfavored facets.	In loan applications, is the relative ratio of false positives to false negatives the same across all age demographics?	<p>The range for binary and multicategory facet labels: <math>(-\infty, +\infty)</math>.</p> <ul style="list-style-type: none"> <li>Positive values occur when the ratio of false positives to false negatives for facet <math>a</math> is greater than that for facet <math>d</math>.</li> <li>Values near zero occur when the ratio of false positives to false negatives for facet <math>a</math> is similar to that for facet <math>d</math>.</li> <li>Negative values occur when the ratio of false positives to false negatives for facet <math>a</math> is less than that for facet <math>d</math>.</li> </ul>

posttraining bias metric	Description	Example question	Interpreting metric values
<a href="#">Conditional Demographic Disparity in Predicted Labels (CDDPL) (p. 1838)</a>	Measures the disparity of predicted labels between the facets as a whole, but also by subgroups.	Do some demographic groups have a larger proportion of rejections for loan application outcomes than their proportion of acceptances?	<p>The range of CDDPL values for binary, multicategory, and continuous outcomes: [-1, +1]</p> <ul style="list-style-type: none"> <li>Positive values indicate an outcomes where facet <math>d</math> is rejected more than accepted.</li> <li>Near zero indicates no demographic disparity on average.</li> <li>Negative values indicate an outcomes where facet <math>a</math> is rejected more than accepted.</li> </ul>
<a href="#">Counterfactual Fliptest (FT) (p. 1839)</a>	Examines each member of facet $d$ and assesses whether similar members of facet $a$ have different model predictions.	Are a group of a specific age demographic, matched closely on all features with another age group, paid on average more than that other age group?"	<p>The range for binary and multicategory facet labels is [-1, +1].</p> <ul style="list-style-type: none"> <li>Positive values occur when the number of unfavorable counterfactual fliptest decisions for the disfavored facet <math>d</math> exceeds the favorable ones.</li> <li>Values near zero occur when the number of unfavorable and favorable counterfactual fliptest decisions balance out.</li> <li>Negative values occur when the number of unfavorable counterfactual fliptest decisions for the disfavored facet <math>d</math> is less than the favorable ones.</li> </ul>

For additional information about posttraining bias metrics, see [A Family of Fairness Measures for Machine Learning in Finance](#).

**more false negatives in the disfavored facet d.**

- Difference in Positive Proportions in Predicted Labels (DPPL) (p. 1830)
- Disparate Impact (DI) (p. 1831)
- Difference in Conditional Acceptance (DCAcc) (p. 1831)
- Difference in Conditional Rejection (DCR) (p. 1832)
- Recall Difference (RD) (p. 1834)
- Difference in Acceptance Rates (DAR) (p. 1835)
- Difference in Rejection Rates (DRR) (p. 1835)
- Accuracy Difference (AD) (p. 1836)
- Treatment Equality (TE) (p. 1837)
- Conditional Demographic Disparity in Predicted Labels (CDDPL) (p. 1838)
- Counterfactual Fliptest (FT) (p. 1839)

## Difference in Positive Proportions in Predicted Labels (DPPL)

The difference in positive proportions in predicted labels (DPPL) metric determines whether the model predicts outcomes differently for each facet. It is defined as the difference between the proportion of positive predictions ( $y' = 1$ ) for facet  $a$  and the proportion of positive predictions ( $y' = 1$ ) for facet  $d$ . For example, if the model predictions grant loans to 60% of a middle-aged group (facet  $a$ ) and 50% other age groups (facet  $d$ ), it might be biased against facet  $d$ . In this example, you need to determine whether the 10% difference is material to a case for bias. A comparison of DPL with DPPL assesses whether bias initially present in the dataset increases or decreases in the model predictions after training.

The formula for the difference in proportions of predicted labels:

$$DPPL = q'_a - q'_d$$

Where:

- $q'_a = n'_a^{(1)} / n_a$  is the predicted proportion of facet  $a$  who get a positive outcome of value 1. In our example, the proportion of a middle-aged facet predicted to get granted a loan. Here  $n'_a^{(1)}$  represents the number of members of facet  $a$  who get a positive predicted outcome of value 1 and  $n_a$  the is number of members of facet  $a$ .
- $q'_d = n'_d^{(1)} / n_d$  is the predicted proportion of facet  $d$  who get a positive outcome of value 1. In our example, a facet of older and younger people predicted to get granted a loan. Here  $n'_d^{(1)}$  represents the number of members of facet  $d$  who get a positive predicted outcome and  $n_d$  the is number of members of facet  $d$ .

If DPPL is close enough to 0, it means that posttraining *demographic parity* has been achieved.

For binary and multicategory facet labels, the normalized DPL values range over the interval  $[-1, 1]$ . For continuous labels, the values vary over the interval  $(-\infty, +\infty)$ .

- Positive DPPL values indicate that facet  $a$  has a higher proportion of predicted positive outcomes when compared with facet  $d$ .

This is referred to as *positive bias*.

- Values of DPPL near zero indicate a more equal proportion of predicted positive outcomes between facets  $a$  and  $d$  and a value of zero indicates perfect demographic parity.
- Negative DPPL values indicate that facet  $d$  has a higher proportion of predicted positive outcomes when compared with facet  $a$ . This is referred to as *negative bias*.

## Disparate Impact (DI)

The difference in positive proportions in the predicted labels metric can be assessed in the form of a ratio.

The comparison of positive proportions in predicted labels metric can be assessed in the form of a ratio instead of as a difference, as it is with the [Difference in Positive Proportions in Predicted Labels \(DPPL\) \(p. 1830\)](#). The disparate impact (DI) metric is defined as the ratio of the proportion of positive predictions ( $y' = 1$ ) for facet  $a$  over the proportion of positive predictions ( $y' = 1$ ) for facet  $d$ . For example, if the model predictions grant loans to 60% of a middle-aged group (facet  $a$ ) and 50% other age groups (facet  $d$ ), then  $DI = .6/.5 = 1.2$ , which indicates a positive bias and an adverse impact on facet  $d$ .

The formula for the ratio of proportions of the predicted labels:

$$DI = q'_d/q'_a$$

Where:

- $q'_a = n'_a^{(1)}/n_a$  is the predicted proportion of facet  $a$  who get a positive outcome of value 1. In our example, the proportion of a middle-aged facet predicted to get granted a loan. Here  $n'_a^{(1)}$  represents the number of members of facet  $a$  who get a positive predicted outcome and  $n_a$  the is number of members of facet  $a$ .
- $q'_d = n'_d^{(1)}/n_d$  is the predicted proportion of facet  $d$  a who get a positive outcome of value 1. In our example, a facet of older and younger people predicted to get granted a loan. Here  $n'_d^{(1)}$  represents the number of members of facet  $d$  who get a positive predicted outcome and  $n_d$  the is number of members of facet  $d$ .

For binary, multiclass facet, and continuous labels, the DI values range over the interval  $[0, \infty)$ .

- Values less than 1 indicate that facet  $a$  has a higher proportion of predicted positive outcomes than facet  $d$ . This is referred to as *positive bias*.
- A value of 1 indicates demographic parity.
- Values greater than 1 indicate that facet  $d$  has a higher proportion of predicted positive outcomes than facet  $a$ . This is referred to as *negative bias*.

## Difference in Conditional Acceptance (DCAcc)

This metric compares the observed labels to the labels predicted by the model and assesses whether this is the same across facets for predicted positive outcomes. This metric comes close to mimicking human bias in that it quantifies how many more positive outcomes a model predicted (labels  $y'$ ) for a certain facet as compared to what was observed in the training dataset (labels  $y$ ). For example, if there were more acceptances (a positive outcome) for loan applications for a middle-aged group (facet  $a$ ) than predicted by the model based on qualifications as compared to the facet containing other age groups (facet  $d$ ), this might indicate potential bias in the way loans were approved.

The formula for the difference in conditional acceptance:

$$DCAcc = c_a - c_d$$

Where:

- $c_a = n_a^{(1)}/n'_a^{(1)}$  is the ratio of the observed number of positive outcomes of value 1 (acceptances) of facet  $a$  to the predicted number of positive outcome (acceptances) for facet  $a$ .
- $c_d = n_d^{(1)}/n'_d^{(1)}$  is the ratio of the observed number of positive outcomes of value 1 (acceptances) of facet  $d$  to the predicted number of predicted positive outcomes (acceptances) for facet  $d$ .

The DCAcc metric can capture both positive and negative biases that reveal preferential treatment based on qualifications. Consider the following instances of age-based bias on loan acceptances.

#### **Example 1: Positive bias**

Suppose we have dataset of 100 middle-aged people (facet *a*) and 50 people from other age groups (facet *d*) who applied for loans, where the model recommended that 60 from facet *a* and 30 from facet *d* be given loans. So the predicted proportions are unbiased with respect to the DPPL metric, but the observed labels show that 70 from facet *a* and 20 from facet *d* were granted loans. In other words, the model granted loans to 17% more from the middle aged facet than the observed labels in the training data suggested ( $70/60 = 1.17$ ) and granted loans to 33% less from other age groups than the observed labels suggested ( $20/30 = 0.67$ ). The calculation of the DCAcc value quantifies this difference between +17% and -33%. The calculation of the DCAcc value gives the following:

$$\text{DCAcc} = 70/60 - 20/30 = 1/2$$

#### **Example 2: Negative bias**

Suppose we have dataset of 100 middle-aged people (facet *a*) and 50 people from other age groups (facet *d*) who applied for loans, where the model recommended that 60 from facet *a* and 30 from facet *d* be given loans. So the predicted proportions are unbiased with respect to the DPPL metric, but the observed labels show that 50 from facet *a* and 40 from facet *d* were granted loans. In other words, the model granted loans to 17% fewer from the middle aged facet than the observed labels in the training data suggested ( $50/60 = 0.83$ ), and granted loans to 33% more from other age groups than the observed labels suggested ( $40/30 = 1.33$ ). The calculation of the DCAcc value quantifies this difference between -17% and +33%. The calculation of the DCAcc value gives the following:

$$\text{DCAcc} = 50/60 - 40/30 = -1/2$$

Note that you can use DCAcc to help you detect potential (unintentional) biases by humans overseeing the model predictions in a human-in-the-loop setting. Assume, for example, that the predictions  $y'$  by the model were unbiased, but the eventual decision is made by a human (possibly with access to additional features) who can alter the model predictions to generate a new and final version of  $y'$ . The additional processing by the human may unintentionally deny loans to a disproportionate number from one facet. DCAcc can help detect such potential biases.

The range of values for differences in conditional acceptance for binary, multicategory facet, and continuous labels is  $(-\infty, +\infty)$ .

- Positive values occur when the ratio of the observed number of acceptances compared to predicted acceptances for facet *a* is higher than the same ratio for facet *d*. These values indicate a possible bias against the qualified applicants from facet *d*. The larger the difference of the ratios, the more extreme the apparent bias.
- Values near zero occur when the ratio of the observed number of acceptances compared to predicted acceptances for facet *a* is similar to the ratio for facet *d*. These values indicate that predicted acceptance rates are consistent with the observed values in the labeled data and that qualified applicants from both facets are being accepted in a similar way.
- Negative values occur when the ratio of the observed number of acceptances compared to predicted acceptances for facet *a* is less than that ratio for facet *d*. These values indicate a possible bias against the qualified applicants from facet *a*. The more negative the difference in the ratios, the more extreme the apparent bias.

## Difference in Conditional Rejection (DCR)

This metric compares the observed labels to the labels predicted by the model and assesses whether this is the same across facets for negative outcomes (rejections). This metric comes close to mimicking human bias, in that it quantifies how many more negative outcomes a model granted (predicted labels  $y'$ ) to a certain facet as compared to what was suggested by the labels in the training dataset (observed

labels  $y$ ). For example, if there were more rejections (a negative outcome) for loan applications for a middle-aged group (facet  $a$ ) than predicted by the model based on qualifications as compared to the facet containing other age groups (facet  $d$ ), this might indicate potential bias in the way loans were rejected.

The formula for the difference in conditional acceptance:

$$DCR = r_d - r_a$$

Where:

- $r_d = n_d^{(0)} / n_d^{(0)}$  is the ratio of the observed number of negative outcomes of value 1 (rejections) of facet  $d$  to the predicted number of negative outcome (rejections) for facet  $d$ .
- $r_a = n_a^{(0)} / n_a^{(0)}$  is the ratio of the observed number of negative outcomes of value 0 (rejections) of facet  $a$  to the predicted number of negative outcome of value 0 (rejections) for facet  $a$ .

The DCR metric can capture both positive and negative biases that reveal preferential treatment based on qualifications. Consider the following instances of age-based bias on loan rejections.

#### **Example 1: Positive bias**

Suppose we have dataset of 100 middle-aged people (facet  $a$ ) and 50 people from other age groups (facet  $d$ ) who applied for loans, where the model recommended that 60 from facet  $a$  and 30 from facet  $d$  be rejected for loans. So the predicted proportions are unbiased by the DPPL metric, but the observed labels show that 50 from facet  $a$  and 40 from facet  $d$  were rejected. In other words, the model rejected loans for 17% fewer from the middle aged facet than the observed labels in the training data suggested ( $50/60 = 0.83$ ), and rejected loans for 33% more from other age groups than the observed labels suggested ( $40/30 = 1.33$ ). The calculation of the DCR value quantifies this difference between -17% and +33%.

$$DCR = 40/30 - 50/60 = 1/2$$

#### **Example 2: Negative bias**

Suppose we have dataset of 100 middle-aged people (facet  $a$ ) and 50 people from other age groups (facet  $d$ ) who applied for loans, where the model recommended that 60 from facet  $a$  and 30 from facet  $d$  be rejected for loans. So the predicted proportions are unbiased by the DPPL metric, but the observed labels show that 70 from facet  $a$  and 20 from facet  $d$  were rejected. In other words, the model rejected loans for 17% more from the middle aged facet than the observed labels in the training data suggested ( $70/60 = 1.17$ ), and rejected loans for 33% fewer from other age groups than the observed labels suggested ( $20/30 = 0.67$ ). The calculation of the DCR value quantifies this difference between 17% and -33%.

$$DCR = 20/30 - 70/60 = -1/2$$

The range of values for differences in conditional rejection for binary, multicategory facet, and continuous labels is  $(-\infty, +\infty)$ .

- Positive values occur when the ratio of the observed number of rejections compared to predicted rejections for facet  $d$  is greater than that ratio for facet  $a$ . These values indicate a possible bias against the qualified applicants from facet  $d$ . The larger the value of DCR metric, the more extreme the apparent bias.
- Values near zero occur when the ratio of the observed number of rejections compared to predicted acceptances for facet  $a$  is similar to the ratio for facet  $d$ . These values indicate that predicted rejections rates are consistent with the observed values in the labeled data and that the qualified applicants from both facets are being rejected in a similar way.
- Negative values occur when the ratio of the observed number of rejections compared to predicted rejections for facet  $d$  is less than that ratio for facet  $a$ . These values indicate a possible bias against the

qualified applicants from facet  $a$ . The larger magnitude of the negative DCR metric, the more extreme the apparent bias.

## Recall Difference (RD)

The recall difference (RD) metric is the difference in recall of the model between the favored facet  $a$  and disfavored facet  $d$ . Any difference in these recalls is a potential form of bias. Recall is the true positive rate (TPR), which measures how often the model correctly predicts the cases that should receive a positive outcome. Recall is perfect for a facet if all of the  $y=1$  cases are correctly predicted as  $y'=1$  for that facet. Recall is greater when the model minimizes false negatives known as the Type II error. For example, how many of the people in two different groups (facets  $a$  and  $d$ ) that should qualify for loans are detected correctly by the model? If the recall rate is high for lending to facet  $a$ , but low for lending to facet  $d$ , the difference provides a measure of this bias against the group belonging to facet  $d$ .

The formula for difference in the recall rates for facets  $a$  and  $d$ :

$$RD = TP_a/(TP_a + FN_a) - TP_d/(TP_d + FN_d) = TPR_a - TPR_d$$

Where:

- $TP_a$  are the true positives predicted for facet  $a$ .
- $FN_a$  are the false negatives predicted for facet  $a$ .
- $TP_d$  are the true positives predicted for facet  $d$ .
- $FN_d$  are the false negatives predicted for facet  $d$ .
- $TPR_a = TP_a/(TP_a + FN_a)$  is the recall for facet  $a$ . or its true positive rate.
- $TPR_d = TP_d/(TP_d + FN_d)$  is the recall for facet  $d$ . or its true positive rate.

For example, consider the following confusion matrices for facets  $a$  and  $d$ .

### Confusion Matrix for the Favored Facet a

Class a predictions	Actual outcome 0	Actual outcome 1	Total
0	20	5	25
1	10	65	75
Total	30	70	100

### Confusion Matrix for the Disfavored Facet d

Class d predictions	Actual outcome 0	Actual outcome 1	Total
0	18	7	25
1	5	20	25
Total	23	27	50

The value of the recall difference is  $RD = 65/70 - 20/27 = 0.93 - 0.74 = 0.19$  which indicates a bias against facet  $d$ .

The range of values for the recall difference between facets  $a$  and  $d$  for binary and multiclass classification is  $[-1, +1]$ . This metric is not available for the case of continuous labels.

- Positive values are obtained when there is higher recall for facet  $a$  than for facet  $d$ . This suggests that the model finds more of the true positives for facet  $a$  than for facet  $d$ , which is a form of bias.
- Values near zero indicate that the recall for facets being compared is similar. This suggests that the model finds about the same number of true positives in both of these facets and is not biased.
- Negative values are obtained when there is higher recall for facet  $d$  than for facet  $a$ . This suggests that the model finds more of the true positives for facet  $d$  than for facet  $a$ , which is a form of bias.

## Difference in Acceptance Rates (DAR)

The difference in acceptance rates (DAR) metric is the difference in the ratios of the true positive (TP) predictions to the observed positives (TP + FP) for facets  $a$  and  $d$ . This metric measures the difference in the precision of the model for predicting acceptances from these two facets. Precision measures the fraction of qualified candidates from the pool of qualified candidates that are identified as such by the model. If the model precision for predicting qualified applicants diverges between the facets, this is a bias and its magnitude is measured by the DAR.

The formula for difference in acceptance rates between facets  $a$  and  $d$ :

$$\text{DAR} = \text{TP}_a / (\text{TP}_a + \text{FP}_a) - \text{TP}_d / (\text{TP}_d + \text{FP}_d)$$

Where:

- $\text{TP}_a$  are the true positives predicted for facet  $a$ .
- $\text{FP}_a$  are the false positives predicted for facet  $a$ .
- $\text{TP}_d$  are the true positives predicted for facet  $d$ .
- $\text{FP}_d$  are the false positives predicted for facet  $d$ .

For example, suppose the model accepts 70 middle-aged applicants (facet  $a$ ) for a loan (predicted positive labels) of whom only 35 are actually accepted (observed positive labels). Also suppose the model accepts 100 applicants from other age demographics (facet  $d$ ) for a loan (predicted positive labels) of whom only 40 are actually accepted (observed positive labels). Then  $\text{DAR} = 35/70 - 40/100 = 0.10$ , which indicates a potential bias against qualified people from the second age group (facet  $d$ ).

The range of values for DAR for binary, multicategory facet, and continuous labels is  $[-1, +1]$ .

- Positive values occur when the ratio of the predicted positives (acceptances) to the observed positive outcomes (qualified applicants) for facet  $a$  is larger than the same ratio for facet  $d$ . These values indicate a possible bias against the disfavored facet  $d$  caused by the occurrence of relatively more false positives in facet  $d$ . The larger the difference in the ratios, the more extreme the apparent bias.
- Values near zero occur when the ratio of the predicted positives (acceptances) to the observed positive outcomes (qualified applicants) for facets  $a$  and  $d$  have similar values indicating the observed labels for positive outcomes are being predicted with equal precision by the model.
- Negative values occur when the ratio of the predicted positives (acceptances) to the observed positive outcomes (qualified applicants) for facet  $d$  is larger than the ratio for facet  $a$ . These values indicate a possible bias against the favored facet  $a$  caused by the occurrence of relatively more false positives in facet  $a$ . The more negative the difference in the ratios, the more extreme the apparent bias.

## Difference in Rejection Rates (DRR)

The difference in rejection rates (DRR) metric is the difference in the ratios of the true negative (TN) predictions to the observed negatives (TN + FN) for facets  $a$  and  $d$ . This metric measures the difference in the precision of the model for predicting rejections from these two facets. Precision measures the fraction of unqualified candidates from the pool of unqualified candidates that are identified as such by

the model. If the model precision for predicting unqualified applicants diverges between the facets, this is a bias and its magnitude is measured by the DRR.

The formula for difference in rejection rates between facets  $a$  and  $d$ :

$$\text{DRR} = \text{TN}_d / (\text{TN}_d + \text{FN}_d) - \text{TN}_a / (\text{TN}_a + \text{FN}_a)$$

Where:

- $\text{TN}_d$  are the true negatives predicted for facet  $d$ .
- $\text{FN}_d$  are the false negatives predicted for facet  $d$ .
- $\text{TN}_a$  are the true negatives predicted for facet  $a$ .
- $\text{FN}_a$  are the false negatives predicted for facet  $a$ .

For example, suppose the model rejects 100 middle-aged applicants (facet  $a$ ) for a loan (predicted negative labels) of whom 80 are actually unqualified (observed negative labels). Also suppose the model accepts 50 applicants from other age demographics (facet  $d$ ) for a loan (predicted negative labels) of whom only 40 are actually unqualified (observed negative labels). Then  $\text{DRR} = 40/50 - 80/100 = 0$ , so no bias is indicated. When both DAR and DRR are zero, it satisfies a condition known as *equalized odds*.

The range of values for DRR for binary, multicategory facet, and continuous labels is  $[-1, +1]$ .

- Positive values occur when the ratio of the predicted negatives (rejections) to the observed negative outcomes (unqualified applicants) for facet  $d$  is larger than the same ratio for facet  $a$ . These values indicate a possible bias against the favored facet  $a$  caused by the occurrence of relatively more false negatives in facet  $a$ . The larger the difference in the ratios, the more extreme the apparent bias.
- Values near zero occur when the ratio of the predicted negatives (rejections) to the observed negative outcomes (unqualified applicants) for facets  $a$  and  $d$  have similar values, indicating the observed labels for negative outcomes are being predicted with equal precision by the model.
- Negative values occur when the ratio of the predicted negatives (rejections) to the observed negative outcomes (unqualified applicants) for facet  $a$  is larger than the ratio for facet  $d$ . These values indicate a possible bias against the disfavored facet  $d$  caused by the occurrence of relatively more false positives in facet  $d$ . The more negative the difference in the ratios, the more extreme the apparent bias.

## Accuracy Difference (AD)

Accuracy difference (AD) metric is the difference between the prediction accuracy for different facets. This metric determines whether the classification by the model is more accurate for one facet than the other. AD indicates whether one facet incurs a greater proportion of Type I and Type II errors. But it cannot differentiate between Type I and Type II errors. For example, the model may have equal accuracy for different age demographics, but the errors may be mostly false positives (Type I errors) for one age-based group and mostly false negatives (Type II errors) for the other.

Also, if loan approvals are made with much higher accuracy for a middle-aged demographic (facet  $a$ ) than for another age-based demographic (facet  $d$ ), either a greater proportion of qualified applicants in the second group are denied a loan (FN) or a greater proportion of unqualified applicants from that group get a loan (FP) or both. This can lead to within group unfairness for the second group, even if the proportion of loans granted is nearly the same for both age-based groups, which is indicated by a DPPL value that is close to zero.

The formula for AD metric is the difference between the prediction accuracy for facet  $a$ ,  $\text{ACC}_a$ , minus that for facet  $d$ ,  $\text{ACC}_d$ :

$$\text{AD} = \text{ACC}_a - \text{ACC}_d$$

Where:

- $ACC_a = (TP_a + TN_a) / (TP_a + TN_a + FP_a + FN_a)$ 
  - $TP_a$  are the true positives predicted for facet  $a$
  - $TN_a$  are the true negatives predicted for facet  $a$
  - $FP_a$  are the false positives predicted for facet  $a$
  - $FN_a$  are the false negatives predicted for facet  $a$
- $ACC_d = (TP_d + TN_d) / (TP_d + TN_d + FP_d + FN_d)$ 
  - $TP_d$  are the true positives predicted for facet  $d$
  - $TN_d$  are the true negatives predicted for facet  $d$
  - $FP_d$  are the false positives predicted for facet  $d$
  - $FN_d$  are the false negatives predicted for facet  $d$

For example, suppose a model approves loans to 70 applicants from facet  $a$  of 100 and rejected the other 30. 10 should not have been offered the loan ( $FP_a$ ) and 60 were approved that should have been ( $TP_a$ ). 20 of the rejections should have been approved ( $FN_a$ ) and 10 were correctly rejected ( $TN_a$ ). The accuracy for facet  $a$  is as follows:

$$ACC_a = (60 + 10) / (60 + 10 + 20 + 10) = 0.7$$

Next, suppose a model approves loans to 50 applicants from facet  $d$  of 100 and rejected the other 50. 10 should not have been offered the loan ( $FP_d$ ) and 40 were approved that should have been ( $TP_d$ ). 40 of the rejections should have been approved ( $FN_d$ ) and 10 were correctly rejected ( $TN_d$ ). The accuracy for facet  $a$  is determined as follows:

$$ACC_d = (40 + 10) / (40 + 10 + 40 + 10) = 0.5$$

The accuracy difference is thus  $AD = ACC_a - ACC_d = 0.7 - 0.5 = 0.2$ . This indicates there is a bias against facet  $d$  as the metric is positive.

The range of values for  $AD$  for binary and multicategory facet labels is  $[-1, +1]$ .

- Positive values occur when the prediction accuracy for facet  $a$  is greater than that for facet  $d$ . It means that facet  $d$  suffers more from some combination of false positives (Type I errors) or false negatives (Type II errors). This means there is a potential bias against the disfavored facet  $d$ .
- Values near zero occur when the prediction accuracy for facet  $a$  is similar to that for facet  $d$ .
- Negative values occur when the prediction accuracy for facet  $d$  is greater than that for facet  $a$ . It means that facet  $a$  suffers more from some combination of false positives (Type I errors) or false negatives (Type II errors). This means there is a bias against the favored facet  $a$ .

## Treatment Equality (TE)

The treatment equality (TE) is the difference in the ratio of false positives to false negatives between facets  $a$  and  $d$ . The main idea of this metric is to assess whether, even if the accuracy across groups is the same, is it the case that errors are more harmful to one group than another? Error rate comes from the total of false positives and false negatives, but the breakdown of these two maybe very different across facets. TE measures whether errors are compensating in the similar or different ways across facets.

The formula for the treatment equality:

$$TE = FP_a/FN_a - FP_d/FN_d$$

Where:

- $FP_a$  are the false positives predicted for facet  $a$ .
- $FN_a$  are the false negatives predicted for facet  $a$ .

- $FP_d$  are the false positives predicted for facet  $d$ .
- $FN_d$  are the false negatives predicted for facet  $d$ .

Note the metric becomes unbounded if  $FN_a$  or  $FN_d$  is zero.

For example, suppose that there are 100 loan applicants from facet  $a$  and 50 from facet  $d$ . For facet  $a$ , 8 were wrongly denied a loan ( $FN_a$ ) and another 6 were wrongly approved ( $FP_a$ ). The remaining predictions were true, so  $TP_a + TN_a = 86$ . For facet  $d$ , 5 were wrongly denied ( $FN_d$ ) and 2 were wrongly approved ( $FP_d$ ). The remaining predictions were true, so  $TP_d + TN_d = 43$ . The ratio of false positives to false negatives equals  $6/8 = 0.75$  for facet  $a$  and  $2/5 = 0.40$  for facet  $d$ . Hence  $TE = 0.75 - 0.40 = 0.35$ , even though both facets have the same accuracy:

$$ACC_a = (86)/(68+ 8 + 6) = 0.86$$

$$ACC_d = (43)/(43 + 5 + 2) = 0.86$$

The range of values for differences in conditional rejection for binary and multiclass facet labels is  $(-\infty, +\infty)$ . The TE metric is not defined for continuous labels. The interpretation of this metric depends on the relative importance of false positives (Type I error) and false negatives (Type II error).

- Positive values occur when the ratio of false positives to false negatives for facet  $a$  is greater than that for facet  $d$ .
- Values near zero occur when the ratio of false positives to false negatives for facet  $a$  is similar to that for facet  $d$ .
- Negative values occur when the ratio of false positives to false negatives for facet  $a$  is less than that for facet  $d$ .

## Conditional Demographic Disparity in Predicted Labels (CDDPL)

The demographic disparity metric (DDPL) determines whether facet  $d$  has a larger proportion of the predicted rejected labels than of the predicted accepted labels. It enables a comparison of difference in predicted rejection proportion and predicted acceptance proportion across facets. This metric is exactly the same as the pretraining CDD metric except that it is computed off the predicted labels instead of the observed ones. This metric lies in the range  $(-1,+1)$ .

The formula for the demographic disparity predictions for labels of facet  $d$  is as follows:

$$DDPL_d = n'_d^{(0)}/n'^{(0)} - n'_d^{(1)}/n'^{(1)} = P_d^R(y'^0) - P_d^A(y'^1)$$

Where:

- $n'^{(0)} = n'_a^{(0)} + n'_d^{(0)}$  is the number of predicted rejected labels for facets  $a$  and  $d$ .
- $n'^{(1)} = n'_a^{(1)} + n'_d^{(1)}$  is the number of predicted accepted labels for facets  $a$  and  $d$ .
- $P_d^R(y'^0)$  is the proportion of predicted rejected labels (value 0) in facet  $d$ .
- $P_d^A(y'^1)$  is the proportion of predicted accepted labels (value 1) in facet  $d$ .

A conditional demographic disparity in predicted labels (CDDPL) metric that conditions DDPL on attributes that define a strata of subgroups on the dataset is needed to rule out Simpson's paradox. The regrouping can provide insights into the cause of apparent demographic disparities for less favored facets. The classic case arose in the case of Berkeley admissions where men were accepted at a higher rate overall than women. But when departmental subgroups were examined, women were shown to have higher admission rates than men by department. The explanation was that women had applied to departments with lower acceptance rates than men had. Examining the subgroup acceptance rates revealed that women were actually accepted at a higher rate than men for the departments with lower acceptance rates.

The CDDPL metric gives a single measure for all of the disparities found in the subgroups defined by an attribute of a dataset by averaging them. It is defined as the weighted average of demographic disparities in predicted labels (DDPL<sub>i</sub>) for each of the subgroups, with each subgroup disparity weighted in proportion to the number of observations it contains. The formula for the conditional demographic disparity in predicted labels is as follows:

$$CDDPL = (1/n) \cdot \sum_i n_i \cdot DDPL_i$$

Where:

- $\sum_i n_i = n$  is the total number of observations and  $n_i$  is the number of observations for each subgroup.
- $DDPL_i = n_i^{(0)}/n^{(0)} - n_i^{(1)}/n^{(1)} = P_i^R(y^0) - P_i^A(y^1)$  is the demographic disparity in predicted labels for the subgroup.

So the demographic disparity for a subgroup in predicted labels (DDPL<sub>i</sub>) are the difference between the proportion of predicted rejected labels and the proportion of predicted accepted labels for each subgroup.

The range of DDPL values for binary, multiclass, and continuous outcomes is [-1, +1].

- +1: when there are no predicted rejection labels for facet  $a$  or subgroup and no predicted acceptances for facet  $d$  or subgroup.
- Positive values indicate there is a demographic disparity in predicted labels as facet  $d$  or subgroup has a larger proportion of the predicted rejected labels than of the predicted accepted labels. The higher the value the greater the disparity.
- Values near zero indicate there is no demographic disparity on average.
- Negative values indicate there is a demographic disparity in predicted labels as facet  $a$  or subgroup has a larger proportion of the predicted rejected labels than of the predicted accepted labels. The lower the value the greater the disparity.
- -1: when there are no predicted rejection labels for facet  $d$  or subgroup and no predicted acceptances for facet  $a$  or subgroup.

## Counterfactual Fliptest (FT)

The fliptest is an approach that looks at each member of facet  $d$  and assesses whether similar members of facet  $a$  have different model predictions. The members of facet  $a$  are chosen to be  $k$ -nearest neighbors of the observation from facet  $d$ . We assess how many nearest neighbors of the opposite group receive a different prediction, where the flipped prediction can go from positive to negative and vice versa.

The formula for the counterfactual fliptest is the difference in the cardinality of two sets divided by the number of members of facet  $d$ :

$$FT = (F^+ - F^-)/n_d$$

Where:

- $F^+$  = is the number of disfavored facet  $d$  members with an unfavorable outcome whose nearest neighbors in favored facet  $a$  received a favorable outcome.
- $F^-$  = is the number of disfavored facet  $d$  members with a favorable outcome whose nearest neighbors in favored facet  $a$  received an unfavorable outcome.
- $n_d$  is the sample size of facet  $d$ .

The range of values for the counterfactual fliptest for binary and multiclass facet labels is [-1, +1]. For continuous labels, we set a threshold to collapse the labels to binary.

- Positive values occur when the number of unfavorable counterfactual fliptest decisions for the disfavored facet  $d$  exceeds the favorable ones.
- Values near zero occur when the number of unfavorable and favorable counterfactual fliptest decisions balance out.
- Negative values occur when the number of unfavorable counterfactual fliptest decisions for the disfavored facet  $d$  is less than the favorable ones.

## Configure an Amazon SageMaker Clarify Processing Jobs for Fairness and Explainability

This topic describes how to configure an Amazon SageMaker Clarify processing job capable of computing bias metrics and feature attributions for explainability. It is implemented using a specialized SageMaker Clarify container image. Instructions are provided for how to locate and download one of these container images. A brief overview of how SageMaker Clarify works is sketched. The parameters needed to configure the processing job and type of analysis are described. Prerequisites are outlined and some advice about compute resources consumed by SageMaker Clarify processing job is provided.

### Topics

- [Prerequisites \(p. 1840\)](#)
- [Getting Started with a SageMaker Clarify Container \(p. 1840\)](#)
- [How It Works \(p. 1842\)](#)
- [Configure a Processing Job Container's Input and Output Parameters \(p. 1842\)](#)
- [Configure the Analysis \(p. 1843\)](#)

## Prerequisites

Before you begin, you need to meet the following prerequisites:

- You need to provide an input dataset as tabular files in CSV or JSONLines format. The input dataset must include a label column for bias analysis. The dataset should be prepared for machine learning with any pre-processing needed, such as data cleaning or feature engineering, already completed.
- You need to provide a model artifact that supports either the CSV or JSONLines file format as one of its content type inputs. For posttraining bias metrics and explainability, we use the dataset to make inferences with the model artifact. Each row minus the label column must be ready to be used as payload for inferences.
- When creating processing jobs with the SageMaker container image, you need the following:
  - Network isolation must be disabled for the processing job.
  - If the model is in a VPC, the processing job must be in the same VPC as the model.
  - The IAM user/role of the caller must have permissions for SageMaker APIs. We recommend that you use the “`arn:aws:iam::aws:policy/AmazonSageMakerFullAccess`” managed policy.

## Getting Started with a SageMaker Clarify Container

Amazon SageMaker provides prebuilt SageMaker Clarify container images that include the libraries and other dependencies needed to compute bias metrics and feature attributions for explainability. This image has been enabled to run SageMaker [Process Data \(p. 664\)](#) in your account.

The image URIs for the containers are in the following form:

```
<ACCOUNT_ID>.dkr.ecr.<REGION_NAME>.amazonaws.com/sagemaker-clarify-processing:1.0
```

For example:

205585389593.dkr.ecr.us-east-1.amazonaws.com/sagemaker-clarify-processing:1.0

The following table lists the addresses by Amazon Region.

### Docker Images for Pretraining Bias Detection

Region	Image address
us-east-1	205585389593.dkr.ecr.us-east-1.amazonaws.com/sagemaker-clarify-processing:1.0
us-east-2	211330385671.dkr.ecr.us-east-2.amazonaws.com/sagemaker-clarify-processing:1.0
us-west-1	740489534195.dkr.ecr.us-west-1.amazonaws.com/sagemaker-clarify-processing:1.0
us-west-2	306415355426.dkr.ecr.us-west-2.amazonaws.com/sagemaker-clarify-processing:1.0
ap-east-1	098760798382.dkr.ecr.ap-east-1.amazonaws.com/sagemaker-clarify-processing:1.0
ap-south-1	452307495513.dkr.ecr.ap-south-1.amazonaws.com/sagemaker-clarify-processing:1.0
ap-northeast-2	263625296855.dkr.ecr.ap-northeast-2.amazonaws.com/sagemaker-clarify-processing:1.0
ap-southeast-1	834264404009.dkr.ecr.ap-southeast-1.amazonaws.com/sagemaker-clarify-processing:1.0
ap-southeast-2	007051062584.dkr.ecr.ap-southeast-2.amazonaws.com/sagemaker-clarify-processing:1.0
ap-northeast-1	377024640650.dkr.ecr.ap-northeast-1.amazonaws.com/sagemaker-clarify-processing:1.0
ca-central-1	675030665977.dkr.ecr.ca-central-1.amazonaws.com/sagemaker-clarify-processing:1.0
eu-central-1	017069133835.dkr.ecr.eu-central-1.amazonaws.com/sagemaker-clarify-processing:1.0
eu-west-1	131013547314.dkr.ecr.eu-west-1.amazonaws.com/sagemaker-clarify-processing:1.0
eu-west-2	440796970383.dkr.ecr.eu-west-2.amazonaws.com/sagemaker-clarify-processing:1.0
eu-west-3	341593696636.dkr.ecr.eu-west-3.amazonaws.com/sagemaker-clarify-processing:1.0
eu-north-1	763603941244.dkr.ecr.eu-north-1.amazonaws.com/sagemaker-clarify-processing:1.0
me-south-1	835444307964.dkr.ecr.me-south-1.amazonaws.com/sagemaker-clarify-processing:1.0

Region	Image address
sa-east-1	520018980103.dkr.ecr.sa-east-1.amazonaws.com/sagemaker-clarify-processing:1.0
af-south-1	811711786498.dkr.ecr.af-south-1.amazonaws.com/sagemaker-clarify-processing:1.0
eu-south-1	638885417683.dkr.ecr.eu-south-1.amazonaws.com/sagemaker-clarify-processing:1.0

## How It Works

A SageMaker processing job used the SageMaker Clarify container at several stages in the lifecycle of the machine learning workflow. You can use the SageMaker Clarify container with your datasets and models to compute the following types of analysis:

- pretraining bias metrics
- posttraining bias metrics
- SHAP values for explainability

You can control which of these analyses are computed when you configure the processing job. For pretraining bias metrics, you need to provide the dataset. You can compute posttraining bias metrics and explainability after your model has been trained by providing the dataset and model name. You must configure the necessary parameters in the form of a JSON configuration file and provide this as an input to the processing job.

After the processing job completes, the result of the analyses is saved in the output location specified in the `ProcessingOutput` parameters of the processing job. You can then download it from there and view the outputs or you can view the results in Studio if you have run a notebook there.

In order to compute posttraining bias metrics and SHAP values, the computation needs to get inferences for the model name provided. To accomplish this, the processing job creates an ephemeral endpoint with the model name, known as a *shadow endpoint*. The processing job deletes the shadow endpoint after the computations are completed.

At a high level, the processing job completes the following steps:

1. Validate inputs and parameters.
2. Create the shadow endpoint.
3. Compute pretraining bias metrics.
4. Compute posttraining bias-metrics.
5. Compute local and global feature attributions.
6. Delete shadow endpoint.
7. Generate output files.

## Configure a Processing Job Container's Input and Output Parameters

The Processing Job requires that you specify the following input parameters: a dataset files with input name "dataset" as S3 object or prefix, and an analysis configuration file with input name "analysis\_config" as an S3 object. The job also requires an output parameter: the output location as an S3 prefix.

You can create and run a processing job with SageMaker [CreateProcessingJob](#) API using the Amazon SDK or CLI or [SageMaker Python SDK](#).

Using SageMaker Python SDK, create a Processor using the SageMaker Clarify container image URI:

```
from sagemaker import clarify
clarify_processor = clarify.SageMakerClarifyProcessor(role=role,
                                                       instance_count=1,
                                                       instance_type='ml.c5.xlarge',
                                                       max_runtime_in_seconds=1200,
                                                       volume_size_in_gb=100)
```

Then, you must also provide the input and output parameters for the SageMakerClarifyProcessor. If you provide the "dataset\_uri", the dataset input is optional.

```
dataset_path = "s3://my_bucket/my_folder/train.csv"
analysis_config_path = "s3://my_bucket/my_folder/analysis_config.json"
analysis_result_path = "s3://my_bucket/my_folder/output"

analysis_config_input = ProcessingInput(
    input_name="analysis_config",
    source=analysis_config_path,
    destination="/opt/ml/processing/input/config",
    s3_data_type="S3Prefix",
    s3_input_mode="File",
    s3_compression_type="None")
dataset_input = ProcessingInput(
    input_name="dataset",
    source=dataset_path,
    destination="/opt/ml/processing/input/data",
    s3_data_type="S3Prefix",
    s3_input_mode="File",
    s3_compression_type="None")
analysis_result_output = ProcessingOutput(
    source="/opt/ml/processing/output",
    destination=analysis_result_path,
    output_name="analysis_result",
    s3_upload_mode="EndOfJob")
```

## Configure the Analysis

Parameters for the analysis configuration must be provided in a JSON file named "analysis\_config.json". The path must be provided in the `ProcessingInput` parameter named "analysis\_config". Examples of analysis configuration files are provided below. In this JSON configuration file, you specify the following parameters:

- "version" – (Optional) Schema version of the configuration file. The latest supported version is used if not provided.
- "dataset\_type" – Format of the dataset. Valid values are "text/csv" for CSV and "application/jsonlines" for JSON Lines.
- "dataset\_uri" – (Optional) Dataset S3 prefix/object URI (if not given as `ProcessingInput`). If it is a prefix, then processing job recursively collects all S3 files under the prefix.
- "headers" – (Optional) A list of column names in the dataset. If the `dataset_type` is "application/jsonlines" and "label" is specified, then the last header shall be the header of label column.
- "label" – (Optional) Target attribute for the model to be used for *bias metrics* specified as a column name or index if the dataset format is CSV or as JSONPath if the dataset format is JSONLines.
- "probability\_threshold" – (Optional) A float value to indicate the threshold to select the binary label in the case of binary classification. The default value is 0.5.

- "features" – (Optional) JSONPath for locating the feature columns for bias metrics if the dataset format is JSONLines.
- "label\_values\_or\_threshold" – (Optional) List of label values or threshold to indicate positive outcome used for bias metrics.
- "facet" – (Optional) A list of features that are sensitive attributes, referred to as facets, to be used for bias metrics in the form of pairs to include the following:
  - "name\_or\_index" – Facet column name or index.
  - "value\_or\_threshold" – (Optional) List of values or threshold that the facet column can take which indicates the sensitive group, i.e. the group that is used to measure bias against. If not provided, bias metrics are computed as one group versus all for every unique value. If the facet column is numeric this threshold value is applied as the lower bound to select the sensitive group.
- "group\_variable" – (Optional) A column name or index to indicate group variable to be used for the *bias metric Conditional Demographic Disparity*.
- "methods" – A list of methods and their parameters for the analyses and reports. If any section is omitted then it is not computed.
  - "pre\_training\_bias" – (Optional) Section on pretraining bias metrics.
    - "methods" – A list of pretraining metrics to be computed.
  - "post\_training\_bias" – (Optional) Section on posttraining bias metrics.
    - "methods" – A list of posttraining metrics to be computed.
  - "shap" – (Optional) Section on SHAP value computation.
    - "baseline" – A list of rows (at least one) or S3 object URI to be used as the baseline dataset (also known as a background dataset) in the Kernel SHAP algorithm. The format should be the same as the dataset format. Each row should contain only the feature columns/values and omit the label column/values
    - "num\_samples" – Number of samples to be used in the Kernel SHAP algorithm. This number determines the size of the generated synthetic dataset to compute the SHAP values.
    - "agg\_method" – Aggregation method for global SHAP values. Valid values are as follows:
      - "mean\_abs" – Mean of absolute SHAP values for all instances.
      - "median" – Median of SHAP values for all instances.
      - "mean\_sq" – Mean of squared SHAP values for all instances.
    - "use\_logit" – (Optional) Boolean value to indicate if logit function is to be applied to the model predictions. The default value is "false". If "use\_logit" is "true" then the SHAP values have log-odds units.
    - "save\_local\_shap\_values" – (Optional) Boolean value to indicate if local SHAP values are to be saved in the output location. Default is "true".
  - "predictor" – (Optional) Section on model parameters, required if "shap" and "post\_training\_bias" sections are present.
    - "model\_name" – Model name (as created by CreateModel API with container mode as SingleModel).
    - "instance\_type" – Instance type for the shadow endpoint.
    - "initial\_instance\_count" – Instance count for the shadow endpoint.
    - "content\_type" – (Optional) The model input format to be used for getting inferences with the shadow endpoint. Valid values are "text/csv" for CSV and "application/jsonlines". The default value is the same as dataset format.
    - "accept\_type" – (Optional) The model *output* format to be used for getting inferences with the shadow endpoint. Valid values are "text/csv" for CSV and "application/jsonlines". The default value is the same as content\_type.
    - "label" – (Optional) Index or JSONPath location in the model output for the target attribute to be used bias metrics. In CSV accept\_type case, if it is not provided then assume that the model output is a single numeric value corresponding to score or probability.

- "probability" – (Optional) Index or JSONPath location in the model output for probabilities or scores to be used for explainability. For example, if the model output is JSONLines with a list of labels and probabilities, then for bias methods, the label that corresponds to the maximum probability is selected for bias computations. For explainability method, currently all probabilities are explained.
- "label\_headers" – (Optional) A list of values that the "label" takes in the dataset that describe which label value each of the scores returned by the model endpoint correspond to. It is used to extract the label value with the highest score as predicted label for bias computations.
- "content\_template" – (Optional) A template string to be used to construct the model input from dataset instances. It is only used when "content\_type" is "application/jsonlines". The template should have one and only one placeholder, \$features, which is replaced by features list at runtime. For example, given "content\_template": "{\"myfeatures\":\$features}", if an instance (no label) is 1,2,3 then model input becomes JSONLine '{"myfeatures": [1,2,3]}'.
- "report" – (Optional) Section on report parameters. A report is generated if this section is present.
- "name" – (Optional) Filename prefix for the report notebook and PDF file. The default name is "report".
- "title" – (Optional) Title string for the report notebook and PDF file. The default title is "SageMaker Analysis Report".

## Example Analysis Configuration JSON File for a CSV Dataset

```
{
  "dataset_type": "text/csv",
  "headers": [
    "feature_0",
    "feature_1",
    "feature_2",
    "feature_3",
    "target"
  ],
  "label": "target",
  "label_values_or_threshold": [1],
  "probability_threshold": 0.7,
  "facet": [
    {
      "name_or_index": "feature_1",
      "value_or_threshold": [1]
    },
    {
      "name_or_index": "feature_2",
      "value_or_threshold": [0.7]
    }
  ],
  "group_variable": "feature_3",
  "methods": {
    "shap": {
      "baseline": [
        [
          "yes",
          3,
          0.9,
          1
        ]
      ],
      "num_samples": 1000,
      "agg_method": "mean",
      "use_logit": true,
      "save_local_shap_values": true
    },
  }
}
```

```

    "pre_training_bias": {
        "methods": "all"
    },
    "post_training_bias": {
        "methods": "all"
    },
    "report": {
        "name": "report",
        "title": "Analysis Report"
    }
},
"predictor": {
    "model_name": "my_model",
    "instance_type": "ml.m5.xlarge",
    "initial_instance_count": 1,
    "content_type": "text/csv",
    "accept_type": "text/csv",
    "label": 0,
    "probability": 1
}
}

```

Model output as CSV is as follows:

```
Current,"[0.028986845165491104, 0.8253824710845947, 0.028993206098675728,  
0.02898673340678215, 0.029557107016444206, 0.0290389321744442, 0.02905467338860035]"
```

Corresponding predictor configuration is as follows:

```

"predictor": {
    ...,
    "accept_type": "text/csv",
    "label": 0,
    "probability": 1,
    ...
}

```

## Example Analysis Configuration JSON File for a JSONLines Dataset

```

{
    "dataset_type": "application/jsonlines",
    "dataset_uri": "s3://my_bucket/my_folder/dataset.jsonl",
    "headers": ["Label", "Feature1", "Feature2"],
    "label": "data.label",
    "features": "data.features.values",
    "facet": [
        {
            "name_or_index": "Feature1",
            "value_or_threshold": [1,5]
        },
        {
            "name_or_index": "Feature2",
            "value_or_threshold": [2,6]
        }
    ],
    "methods": {
        "shap": {
            "baseline": [
                {"data":{"features":{"values":[9,10]},"label":0}},
                {"data":{"features":{"values":[11,12]},"label":1}}
            ]
        }
    }
}

```

```
        },
        "predictor": {
            "model_name": "my_jsonl_model",
            "instance_type": "ml.m5.xlarge",
            "initial_instance_count": 1
        }
    }
```

Dataset as S3 prefix is as follows:

```
"dataset_uri": "s3://my_bucket/my_folder"
```

Dataset as S3 object is as follows:

```
"dataset_uri": "s3://my_bucket/my_folder/train.csv"
```

Baseline as S3 object is as follows:

```
"baseline": "s3://my_bucket/my_folder/baseline.csv"
```

Model output as JSONLines is as follows:

```
{"predicted_label": "Current", "score": "[0.028986845165491104, 0.8253824710845947,
0.028993206098675728, 0.02898673340678215, 0.029557107016444206, 0.0290389321744442,
0.02905467338860035]"}
```

Corresponding predictor configuration is as follows:

```
"predictor": {
    ...
    "accept_type": "application/jsonlines",
    "label": "predicted_label",
    "probability": "score",
    ...
}
```

## Run SageMaker Clarify Processing Jobs for Bias Analysis and Explainability

You use SageMaker Clarify processing jobs to analyze potential sources of bias in your training data and to check your trained model for bias. For the procedure to analyze the data in SageMaker Studio, see [Generate Reports for Bias in Pretraining Data in SageMaker Studio \(p. 572\)](#). The focus here is on posttraining bias metric and SHAP values for explainability. Model predictions can be a source of bias (for example, if they make predictions that more frequently produce a negative result for one group than another). SageMaker Clarify is integrated with SageMaker Experiments so that after a model has been trained, you can identify attributes you would like to check for bias (for example, income). SageMaker runs a set of algorithms to check the trained model and provides you with a visual report on the different types of bias for each attribute, such as whether high-income earners receive more positive predictions compared to low-income earners.

### Topics

- [Compute Resources Required for SageMaker Clarify Processing Jobs \(p. 1848\)](#)
- [Running the Processing Job \(p. 1848\)](#)
- [Get the Analysis Results \(p. 1849\)](#)

## Compute Resources Required for SageMaker Clarify Processing Jobs

Take the following into consideration when determining the compute resources you need to run SageMaker Clarify processing jobs:

- Processing jobs can take several minutes or more to complete.
- Computing explanations can be more time intensive than the actual inference. This includes the time to launch compute resources.
- Computing explanations can be more compute intensive than the actual inference. Review and monitor the charges you may incur from using SageMaker resources. For more information, see [Amazon SageMaker Pricing](#).

## Running the Processing Job

For an example notebook with instructions on how to run a SageMaker Clarify processing job in Studio to detect posttraining model bias, see [Explainability and bias detection with Amazon SageMaker Clarify](#).

If you need instructions on how to open a notebook in Amazon SageMaker Studio, see [Create or Open an Amazon SageMaker Studio Notebook \(p. 79\)](#). The following code samples are taken from the example notebook listed previously.

After you have trained your model, instantiate the SageMaker Clarify processor using the following command:

```
from sagemaker import clarify
clarify_processor = clarify.SageMakerClarifyProcessor(role=role,
                                                       instance_count=1,
                                                       instance_type='ml.c4.xlarge',
                                                       sagemaker_session=session)
```

Next, configure the input dataset, where to store the output, the label column targeted with a `DataConfig` object, specify information about your trained model with `ModelConfig`, and provide information about the formats of your predictions with `ModelPredictedLabelConfig`.

```
bias_report_output_path = 's3://{}//{}//clarify-bias'.format(bucket, prefix)
bias_data_config = clarify.DataConfig(s3_data_input_path=train_uri,
                                      s3_output_path=bias_report_output_path,
                                      label='Target',
                                      headers=training_data.columns.to_list(),
                                      dataset_type='text/csv')

model_config = clarify.ModelConfig(model_name=model_name,
                                    instance_type='ml.c5.xlarge',
                                    instance_count=1,
                                    accept_type='text/csv')

predictions_config = clarify.ModelPredictedLabelConfig(probability_threshold=0.8)
```

Use `BiasConfig` to provide information on which columns contain the facets (sensitive groups, `Sex`), what the sensitive features (`facet_values_or_threshold`) might be, and what the desirable outcomes are (`label_values_or_threshold`).

```
bias_config = clarify.BiasConfig(label_values_or_threshold=[1],
                                 facet_name='Sex',
                                 facet_values_or_threshold=[0])
```

You can run both the pretraining and posttraining analysis in the processing job at the same time with `run_bias()`.

```
clarify_processor.run_bias(data_config=bias_data_config,
                           bias_config=bias_config,
                           model_config=model_config,
                           model_predicted_label_config=predictions_config,
                           pre_training_methods='all',
                           post_training_methods='all')
```

View the results in Studio or download them from the `bias_report_output_path` S3 bucket.

## Get the Analysis Results

After the processing job is finished, you can download the output files to inspect or visualize the results in Studio. The output directory contains the following files:

- `analysis.json` – Bias metrics and SHAP values in JSON format.
- `report.ipynb` – Static notebook to visualize the bias metrics and SHAP values.
- `explanations_shap/out.csv` – Local (per-instance) SHAP values for each row in the dataset in the same format as input dataset. On each row, the output file contains SHAP values for each feature and the predicted label.

In the analysis JSON file, the bias metrics and SHAP values are organized into three separate sections.

```
{
  "explanations": { . . . }
  "pre_training_bias_metrics": { . . . }
  "post_training_bias_metrics": { . . . }
}
```

SHAP values are in the “`explanations`” section. Values correspond to the global SHAP value for each feature column.

```
"explanations": {
    "kernel_shap": {
        "label0": {
            "global_shap_values": {
                "feature_0": 0.022486410860333206,
                "feature_1": 0.007381025261958729,
                "feature_2": 0.006843906804137847
            },
            "expected_value": 0.508233428001
        }
    }
}
```

The bias metrics are in the pretraining and posttraining bias metrics sections.

```
{
  "post_training_bias_metrics": {
    "label": "target",
    "label_value_or_threshold": "1",
    "facets": {
      "feature_2": [
        {
          "value_or_threshold": "1",
          "metrics": [

```

```
{  
    "name": "DI",  
    "description": "Disparate Impact (DI)",  
    "value": 0.711340206185567  
},  
{  
    "name": "DCR",  
    "description": "Difference in Conditional Rejections (DCR)",  
    "value": -0.34782608695652184  
}  
{ . . . }  
]  
}  
}  
}
```

For more information on bias metrics and SHAP values and how to interpret them, see the [Amazon AI Fairness and Explainability Whitepaper](#).

A bar chart of top SHAP values and tables with the bias metrics are provided in the report notebook.

## Troubleshoot SageMaker Clarify Processing Jobs

If you encounter failures with SageMaker Clarify processing jobs, consult the following scenarios to help identify the issue.

### Note

The failure reason and exit message are intended to contain descriptive messages and exceptions, if encountered, during the run. One common reason is invalid or missing parameters. If you encounter unclear, confusing, or misleading messages or are unable to find a solution, submit feedback.

### Topics

- [Processing job fails to finish \(p. 1850\)](#)
- [Processing job finishes without results and you get a CloudWatch warning message \(p. 1851\)](#)
- [Error message for invalid analysis configuration \(p. 1851\)](#)
- [Bias metric computation fails for several or all metrics \(p. 1851\)](#)
- [Mismatch between analysis config and dataset/model input/output \(p. 1851\)](#)
- [Model returns 500 Internal Server Error or container falls back to per-record predictions due to model error \(p. 1852\)](#)
- [Execution role is invalid \(p. 1852\)](#)
- [Failed to download data \(p. 1852\)](#)
- [Could not connect to SageMaker \(p. 1852\)](#)

## Processing job fails to finish

If the processing job fails to finish, you can try the following:

- Inspect the job logs directly in the notebook where you ran the job in. The job logs are located in the output of the notebook cell where you initiated the run.
- Inspect the job logs in CloudWatch.
- Add the following line in your notebook to describe the last processing job and look for the failure reason and exit message:
  - `clarify_processor.jobs[-1].describe()`

- Execute the following Amazon CLI command to describe the processing job and look for the failure reason and exit message:
  - `aws sagemaker describe-processing-job --processing-job-name <processing-job-id>`

## Processing job finishes without results and you get a CloudWatch warning message

If the processing job finishes but no results are found and a warning message is found in the CloudWatch logs that says "Signal 15 received, cleaning up", this is an indication that the job was stopped either due to a customer request that called the `StopProcessingJob` API or that the job ran out of time allotted for its completion. In the later case, check the maximum runtime in the job configuration (`max_runtime_in_seconds`) and increase it as needed.

## Error message for invalid analysis configuration

- If you get the error message "Unable to load analysis configuration as JSON.", this means that the analysis configuration input file for the processing job does not contain a valid JSON object. Check the validity of the JSON object using a JSON linter.
- If you get the error message "Analysis configuration schema validation error.", this means that the analysis configuration input file for the processing job contains unknown fields or invalid types for some field values. Review the configuration parameters in the file and cross-check them with the parameters listed in the [configuration specification](#) file.

## Bias metric computation fails for several or all metrics

If you receive one of the following error messages "No Label values are present in the predicted Label Column, Positive Predicted Index Series contains all False values." or "Predicted Label Column series datatype is not the same as Label Column series.", try the following:

- Check that the correct dataset is being used.
- Check whether the dataset size is too small; whether, for example, it contains only a few rows. This may cause the model outputs to have the same value or the data type is inferred incorrectly.
- Check if the label or facet is treated as continuous or categorical. SageMaker Clarify uses heuristics to determine the [DataType](#). For post-training bias metrics, the data type returned by the model may not match what is in the dataset or SageMaker Clarify may not be able to transform it correctly.
- In the bias report, you should see a single value for categorical columns or an interval for continuous columns.
- For example, if a column has values 0.0 and 1.0 as floats, it will be treated as continuous even if there are too few unique values.

## Mismatch between analysis config and dataset/model input/output

- Check that the baseline format in the analysis config is the same as dataset format.
- If you receive the error message "Could not convert string to float.", check that the format is correctly specified. It could also indicate that the model predictions have a different format than the label column or it could indicate that the configuration for the label or probabilities is incorrect.
- If you receive the error message "Unable to locate the facet." or "Headers must contain label." or "Headers in config do not match with the number of columns in the dataset." or "Feature names not found.", check that the headers match the columns.

- If you receive the error message "Data must contain features.", check the content template for JSONLines and compare it with the dataset sample if available.

## Model returns 500 Internal Server Error or container falls back to per-record predictions due to model error

If you receive the error message "Fallback to per-record prediction because of model error.", this could indicate that model cannot handle the batch size, or be throttled, or just does not accept the input passed by the container due to serialization problems. You should review the CloudWatch logs for the SageMaker endpoint and look for error messages or tracebacks. For model throttling cases, it may help to use a different instance type or increasing the number of instances for the endpoint.

## Execution role is invalid

This indicates that the role provided is incorrect or missing required permissions. Check the role and its permissions that were used to configure the processing job and verify the permission and trust policy for the role.

## Failed to download data

This indicates that job inputs could not be downloaded for the job to start. Check the bucket name and permissions for the dataset and the configuration inputs.

## Could not connect to SageMaker

This indicates that the job could not reach SageMaker service endpoints. Check the network configuration settings for the processing job and verify VPC configuration.

# Model Explainability

Amazon SageMaker Clarify provides tools to help explain how machine learning (ML) models make predictions. These tools can help ML modelers and developers and other internal stakeholders understand model characteristics as a whole prior to deployment and to debug predictions provided by the model after it's deployed. Transparency about how ML models arrive at their predictions is also critical to consumers and regulators who need to trust the model predictions if they are going to accept the decisions based on them. SageMaker Clarify uses a model-agnostic feature attribution approach, which you can use to understand why a model made a prediction after training and to provide per-instance explanation during inference. The implementation includes a scalable and efficient implementation of [SHAP](#), based on the concept of a Shapley value from the field of cooperative game theory that assigns each feature an importance value for a particular prediction.

What is the function of an explanation in the machine learning context? An explanation can be thought of as the answer to a *Why question* that helps humans understand the cause of a prediction. In the context of an ML model, you might be interested in answering questions such as:

- "Why did the model predict a negative outcome such as a loan rejection for a given applicant?"
- "How does the model make predictions?"
- "Why did the model make an incorrect prediction?"
- "Which features have the largest influence on the behavior of the model?"

You can use explanations for auditing and meeting regulatory requirements, building trust in the model and supporting human decision-making, and debugging and improving model performance.

The need to satisfy the demands for human understanding about the nature and outcomes of ML inference is key to the sort of explanation needed. Research from philosophy and cognitive science disciplines has shown that people care especially about contrastive explanations, or explanations of why an event X happened instead of some other event Y that did not occur. Here, X could be an unexpected or surprising event that happened and Y corresponds to an expectation based on their existing mental model referred to as a *baseline*. Note that for the same event X, different people might seek different explanations depending on their point of view or mental model Y. In the context of explainable AI, you can think of X as the example being explained and Y as a baseline that is typically chosen to represent an uninformative or average example in the dataset. Sometimes, the baseline might be implicit as with an image with all pixels of the same color in the case of ML models for images.

### Topics

- [Feature Attributions that Use Shapley Values \(p. 1853\)](#)
- [SHAP Baselines for Explainability \(p. 1854\)](#)
- [Create Feature Attribute Baselines and Explainability Reports \(p. 1854\)](#)

## Feature Attributions that Use Shapley Values

SageMaker Clarify provides feature attributions based on the concept of [Shapley value](#). You can use Shapley values to determine the contribution that each feature made to model predictions. These attributions can be provided for specific predictions and at a global level for the model as a whole. For example, if you used an ML model for college admissions, the explanations could help determine whether the GPA or the SAT score was the feature most responsible for the model's predictions, and then you can determine how responsible each feature was for determining an admission decision about a particular student.

SageMaker Clarify has taken the concept of Shapley values from game theory and deployed it in a machine learning context. The Shapley value provides a way to quantify the contribution of each player to a game, and hence the means to distribute the total gain generated by a game to its players based on their contributions. In this machine learning context, SageMaker Clarify treats the prediction of the model on a given instance as the *game* and the features included in the model as the *players*. For a first approximation, you might be tempted to determine the marginal contribution or effect of each feature by quantifying the result of either *dropping* that feature from the model or *dropping* all other features from the model. However, this approach does not take into account that features included in a model are often not independent from each other. For example, if two features are highly correlated, dropping either one of the features might not alter the model prediction significantly.

To address these potential dependencies, the Shapley value requires that the outcome of each possible combination (or coalition) of features must be considered to determine the importance of each feature. Given  $d$  features, there are  $2^d$  such possible feature combinations, each corresponding to a potential model. To determine the attribution for a given feature  $f$ , consider the marginal contribution of including  $f$  in all feature combinations (and associated models) that do not contain  $f$ , and take the average. It can be shown that Shapley value is the unique way of assigning the contribution or importance of each feature that satisfies certain desirable properties. In particular, the sum of Shapley values of each feature corresponds to the difference between the predictions of the model and a dummy model with no features. However, even for reasonable values of  $d$ , say 50 features, it is computationally prohibitive and impractical to train  $2^d$  possible models. As a result, SageMaker Clarify needs to make use of various approximation techniques. For this purpose, SageMaker Clarify uses SHapley Additive exPlanations (SHAP), which incorporates such approximations and devised a scalable and efficient implementation of the Kernel SHAP algorithm through additional optimizations.

For additional information on Shapley values, see [A Unified Approach to Interpreting Model Predictions](#).

## SHAP Baselines for Explainability

As noted earlier, explanations are typically contrastive (that is, they account for deviations from a baseline). As a result, for the same model prediction, you can expect to get different explanations with respect to different baselines so your choice of a baseline is crucial. In an ML context, the baseline corresponds to a hypothetical instance that can be either *uninformative* or *informative*. During the computation of Shapley values, SageMaker Clarify generates several new instances between the baseline and the given instance, in which the absence of a feature is modeled by setting the feature value to that of the baseline and the presence of a feature is modeled by setting the feature value to that of the given instance. Thus, the absence of all features corresponds to the baseline and the presence of all features corresponds to the given instance.

How can you choose good baselines? Often it is desirable to select a baseline with very low information content. For example, you can construct an average instance from the training dataset by taking either the median or average for numerical features and the mode for categorical features. For the college admissions example, you might be interested in explaining why a particular applicant was accepted as compared to a baseline acceptances based on an average applicant.

Alternatively, you can choose to generate explanations with respect to informative baselines. For the college admissions scenario, you might want to explain why a particular applicant was rejected when compared with other applicants from similar demographic backgrounds. In this case, you can choose a baseline that represents the applicants of interest, namely those from a similar demographic background. Thus, you can use informative baselines to concentrate the analysis on the specific aspects of a particular model prediction. You can isolate the features for assessment by setting demographic attributes and other features that you can't act on to the same value as in the given instance.

## Create Feature Attribute Baselines and Explainability Reports

For an example notebook with instructions on how to run a SageMaker Clarify processing job in Studio that creates explanations for its predictions relative to a baseline, see [Explainability and bias detection with Amazon SageMaker Clarify](#).

If you need instructions on how to open a notebook in Amazon SageMaker Studio, see [Create or Open an Amazon SageMaker Studio Notebook \(p. 79\)](#). The following code examples are taken from the example notebook listed previously. This section discusses the code related to the use of Shapley values to provide reports that compare the relative contributions each feature made the predictions.

Use SHAPConfig to create the baseline. In this example, the mean\_abs is the mean of absolute SHAP values for all instances, specified as the baseline. You use DataConfig to configure the target variable, data input and output paths, and their formats.

```
shap_config = clarify.SHAPConfig(baseline=[test_features.iloc[0].values.tolist()],
                                 num_samples=15,
                                 agg_method='mean_abs')

explainability_output_path = 's3://{}//{}clarify-explainability'.format(bucket, prefix)
explainability_data_config = clarify.DataConfig(s3_data_input_path=train_uri,
                                               s3_output_path=explainability_output_path,
                                               label='Target',
                                               headers=training_data.columns.to_list(),
                                               dataset_type='text/csv')
```

Then run the explainability job.

```
clarify_processor.run_explainability(data_config=explainability_data_config,
```

```
model_config=model_config,  
explainability_config=shap_config)
```

View the results in Studio or download them from the `explainability_output_path` S3 bucket.

## Incremental Training in Amazon SageMaker

Over time, you might find that a model generates inference that are not as good as they were in the past. With incremental training, you can use the artifacts from an existing model and use an expanded dataset to train a new model. Incremental training saves both time and resources.

Use incremental training to:

- Train a new model using an expanded dataset that contains an underlying pattern that was not accounted for in the previous training and which resulted in poor model performance.
- Use the model artifacts or a portion of the model artifacts from a popular publicly available model in a training job. You don't need to train a new model from scratch.
- Resume a training job that was stopped.
- Train several variants of a model, either with different hyperparameter settings or using different datasets.

For more information about training jobs, see [Train a Model with Amazon SageMaker \(p. 8\)](#).

You can train incrementally using the SageMaker console or the [Amazon SageMaker Python SDK](#).

### Important

Only three built-in algorithms currently support incremental training: [Object Detection Algorithm \(p. 1479\)](#), [Image Classification Algorithm \(p. 1399\)](#), and [Semantic Segmentation Algorithm \(p. 1502\)](#).

### Topics

- [Perform Incremental Training \(Console\) \(p. 1855\)](#)
- [Perform Incremental Training \(API\) \(p. 1857\)](#)

## Perform Incremental Training (Console)

To complete this procedure, you need:

- The Amazon Simple Storage Service (Amazon S3) bucket URI where you've stored the training data.
- The S3 bucket URI where you want to store the output of the job.
- The Amazon Elastic Container Registry path where the training code is stored. For more information, see [Docker Registry Paths and Example Code \(p. 725\)](#).
- The URL of the S3 bucket where you've stored the model artifacts that you want to use in incremental training. To find the URL for the model artifacts, see the details page of the training job used to create the model. To find the details page, in the SageMaker console, choose **Inference**, choose **Models**, and then choose the model.

To restart a stopped training job, use the URL to the model artifacts that are stored in the details page as you would with a model or a completed training job.

### To perform incremental training (console)

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.

2. In the navigation pane, choose **Training**, then choose **Training jobs**.
3. Choose **Create training job**.
4. Provide a name for the training job. The name must be unique within an Amazon Region in an Amazon account. The training job name must have 1 to 63 characters. Valid characters: a-z, A-Z, 0-9, and . : + = @ \_ % - (hyphen).
5. Choose the algorithm that you want to use. For information about algorithms, see [Use Amazon SageMaker Built-in Algorithms \(p. 718\)](#).
6. (Optional) For **Resource configuration**, either leave the default values or increase the resource consumption to reduce computation time.
  - a. (Optional) For **Instance type**, choose the ML compute instance type that you want to use. In most cases, **ml.m4.xlarge** is sufficient.
  - b. For **Instance count**, use the default, 1.
  - c. (Optional) For **Additional volume per instance (GB)**, choose the size of the ML storage volume that you want to provision. In most cases, you can use the default, 1. If you are using a large dataset, use a larger size.
7. Provide information about the input data for the training dataset.
  - a. For **Channel name**, either leave the default (**train**) or enter a more meaningful name for the training dataset, such as **expanded-training-dataset**.
  - b. For **InputMode**, choose **File**. For incremental training, you need to use file input mode.
  - c. For **S3 data distribution type**, choose **FullyReplicated**. This causes each ML compute instance to use a full replicate of the expanded dataset when training incrementally.
  - d. If the expanded dataset is uncompressed, set the **Compression type** to **None**. If the expanded dataset is compressed using Gzip, set it to **Gzip**.
  - e. (Optional) If you are using File input mode, leave **Content type** empty. For Pipe input mode, specify the appropriate MIME type. **Content type** is the multipurpose internet mail extension (MIME) type of the data.
  - f. For **Record wrapper**, if the dataset is saved in RecordIO format, choose **RecordIO**. If your dataset is not saved as a RecordIO formatted file, choose **None**.
  - g. For **S3 data type**, if the dataset is stored as a single file, choose **S3Prefix**. If the dataset is stored as several files in a folder, choose **Manifest**.
  - h. For **S3 location**, provide the URL to the path where you stored the expanded dataset.
  - i. Choose **Done**.
8. To use model artifacts in a training job, you need to add a new channel and provide the needed information about the model artifacts.
  - a. For **Input data configuration**, choose **Add channel**.
  - b. For **Channel name**, enter **model1** to identify this channel as the source of the model artifacts.
  - c. For **InputMode**, choose **File**. Model artifacts are stored as files.
  - d. For **S3 data distribution type**, choose **FullyReplicated**. This indicates that each ML compute instance should use all of the model artifacts for training.
  - e. For **Compression type**, choose **None** because we are using a model for the channel.
  - f. Leave **Content type** empty. Content type is the multipurpose internet mail extension (MIME) type of the data. For model artifacts, we leave it empty.
  - g. Set **Record wrapper** to **None** because model artifacts are not stored in RecordIO format.
  - h. For **S3 data type**, if you are using a built-in algorithm or an algorithm that stores the model as a single file, choose **S3Prefix**. If you are using an algorithm that stores the model as several files, choose **Manifest**.
  - i. For **S3 location**, provide the URL to the path where you stored the model artifacts. Typically, the model is stored with the name **model1.tar.gz**. To find the URL for the model artifacts, in

- the navigation pane, choose **Inference**, then choose **Models**. From the list of models, choose a model to display its details page. The URL for the model artifacts is listed under **Primary container**.
- j. Choose **Done**.
  9. For **Output data configuration**, provide the following information:
    - a. For **S3 location**, type the path to the S3 bucket where you want to store the output data.
    - b. (Optional) For **Encryption key**, you can add your Amazon Key Management Service (Amazon KMS) encryption key to encrypt the output data at rest. Provide the key ID or its Amazon Resource Number (ARN). For more information, see [KMS-Managed Encryption Keys](#).
  10. (Optional) For **Tags**, add one or more tags to the training job. A **tag** is metadata that you can define and assign to Amazon resources. In this case, you can use tags to help you manage your training jobs. A tag consists of a key and a value, which you define. For example, you might want to create a tag with **Project** as a key and a value referring to a project that is related to the training job, such as **Home value forecasts**.
  11. Choose **Create training job**. SageMaker creates and runs training job.

After the training job has completed, the newly trained model artifacts are stored under the **S3 output path** that you provided in the **Output data configuration** field. To deploy the model to get predictions, see [Step 5: Deploy the Model to Amazon EC2 \(p. 63\)](#).

## Perform Incremental Training (API)

This example shows how to use SageMaker APIs to train a model using the SageMaker image classification algorithm and the [Caltech 256 Image Dataset](#), then train a new model using the first one. It uses Amazon S3 for input and output sources. Please see the [incremental training sample notebook](#) for more details on using incremental training.

### Note

In this example we used the original datasets in the incremental training, however you can use different datasets, such as ones that contain newly added samples. Upload the new datasets to S3 and make adjustments to the `data_channels` variable used to train the new model.

Get an Amazon Identity and Access Management (IAM) role that grants required permissions and initialize environment variables:

```
import sagemaker
from sagemaker import get_execution_role

role = get_execution_role()
print(role)

sess = sagemaker.Session()

bucket=sess.default_bucket()
print(bucket)
prefix = 'ic-incr-training'
```

Get the training image for the image classification algorithm:

```
from sagemaker.amazon.amazon_estimator import get_image_uri

training_image = get_image_uri(sess.boto_region_name, 'image-classification',
    repo_version="latest")
#Display the training image
print (training_image)
```

Download the training and validation datasets, then upload them to Amazon Simple Storage Service (Amazon S3):

```

import os
import urllib.request
import boto3

# Define a download function
def download(url):
    filename = url.split('/')[-1]
    if not os.path.exists(filename):
        urllib.request.urlretrieve(url, filename)

# Download the caltech-256 training and validation datasets
download('http://data.mxnet.io/data/caltech-256/caltech-256-60-train.rec')
download('http://data.mxnet.io/data/caltech-256/caltech-256-60-val.rec')

# Create four channels: train, validation, train_lst, and validation_lst
s3train = 's3://{}//{}//train/'.format(bucket, prefix)
s3validation = 's3://{}//{}//validation/'.format(bucket, prefix)

# Upload the first files to the train and validation channels
!aws s3 cp caltech-256-60-train.rec $s3train --quiet
!aws s3 cp caltech-256-60-val.rec $s3validation --quiet

```

Define the training hyperparameters:

```

# Define hyperparameters for the estimator
hyperparams = { "num_layers": "18",
                "resize": "32",
                "num_training_samples": "50000",
                "num_classes": "10",
                "image_shape": "3,28,28",
                "mini_batch_size": "128",
                "epochs": "3",
                "learning_rate": "0.1",
                "lr_scheduler_step": "2,3",
                "lr_scheduler_factor": "0.1",
                "augmentation_type": "crop_color",
                "optimizer": "sgd",
                "momentum": "0.9",
                "weight_decay": "0.0001",
                "beta_1": "0.9",
                "beta_2": "0.999",
                "gamma": "0.9",
                "eps": "1e-8",
                "top_k": "5",
                "checkpoint_frequency": "1",
                "use_pretrained_model": "0",
                "model_prefix": " " }

```

Create an estimator object and train the first model using the training and validation datasets:

```

# Fit the base estimator
s3_output_location = 's3://{}//{}//output'.format(bucket, prefix)
ic = sagemaker.estimator.Estimator(training_image,
                                     role,
                                     instance_count=1,
                                     instance_type='ml.p2.xlarge',
                                     volume_size=50,
                                     max_run=360000,
                                     input_mode='File',

```

```
        output_path=s3_output_location,
        sagemaker_session=sess,
        hyperparameters=hyperparams)

train_data = sagemaker.inputs.TrainingInput(s3train, distribution='FullyReplicated',
                                             content_type='application/x-recordio',
                                             s3_data_type='S3Prefix')
validation_data = sagemaker.inputs.TrainingInput(s3validation,
                                                 distribution='FullyReplicated',
                                                 content_type='application/x-recordio',
                                                 s3_data_type='S3Prefix')

data_channels = {'train': train_data, 'validation': validation_data}

ic.fit(inputs=data_channels, logs=True)
```

To use the model to incrementally train another model, create a new estimator object and use the model artifacts (`ic.model_data`, in this example) for the `model_uri` input argument:

```
# Given the base estimator, create a new one for incremental training
incr_ic = sagemaker.estimator.Estimator(training_image,
                                         role,
                                         instance_count=1,
                                         instance_type='ml.p2.xlarge',
                                         volume_size=50,
                                         max_run=360000,
                                         input_mode='File',
                                         output_path=s3_output_location,
                                         sagemaker_session=sess,
                                         hyperparameters=hyperparams,
                                         model_uri=ic.model_data) # This parameter will
                                         ingest the previous job's model as a new channel
incr_ic.fit(inputs=data_channels, logs=True)
```

After the training job has completed, the newly trained model artifacts are stored under the `S3_output_path` that you provided in `Output_path`. To deploy the model to get predictions, see [Step 5: Deploy the Model to Amazon EC2 \(p. 63\)](#).

## Managed Spot Training in Amazon SageMaker

Amazon SageMaker makes it easy to train machine learning models using managed Amazon EC2 Spot instances. Managed spot training can optimize the cost of training models up to 90% over on-demand instances. SageMaker manages the Spot interruptions on your behalf.

Managed Spot Training uses Amazon EC2 Spot instance to run training jobs instead of on-demand instances. You can specify which training jobs use spot instances and a stopping condition that specifies how long SageMaker waits for a job to run using Amazon EC2 Spot instances. Metrics and logs generated during training runs are available in CloudWatch.

Amazon SageMaker automatic model tuning, also known as hyperparameter tuning, can use managed spot training. For more information on automatic model training, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

Spot instances can be interrupted, causing jobs to take longer to start or finish. You can configure your managed spot training job to use checkpoints. SageMaker copies checkpoint data from a local path to Amazon S3. When the job is restarted, SageMaker copies the data from Amazon S3 back into the local path. The training job can then resume from the last checkpoint instead of restarting. For more information about checkpointing, see [Use Checkpoints in Amazon SageMaker \(p. 1860\)](#).

**Note**

Unless your training job will complete quickly, we recommend you use checkpointing with managed spot training. SageMaker built-in algorithms and marketplace algorithms that do not checkpoint are currently limited to a `MaxWaitTimeInSeconds` of 3600 seconds (60 minutes).

**Topics**

- [Using Managed Spot Training \(p. 1860\)](#)
- [Managed Spot Training Lifecycle \(p. 1860\)](#)

## Using Managed Spot Training

To use managed spot training, create a training job. Set `EnableManagedSpotTraining` to `True` and specify the `MaxWaitTimeInSeconds`. `MaxWaitTimeInSeconds` must be larger than `MaxRuntimeInSeconds`. For more information about creating a training job, see [DescribeTrainingJob](#).

You can calculate the savings from using managed spot training using the formula  $(1 - (\text{BillableTimeInSeconds} / \text{TrainingTimeInSeconds})) * 100$ . For example, if `BillableTimeInSeconds` is 100 and `TrainingTimeInSeconds` is 500, the savings is 80%.

## Managed Spot Training Lifecycle

You can monitor a training job using `TrainingJobStatus` and `SecondaryStatus` returned by [DescribeTrainingJob](#). The list below shows how `TrainingJobStatus` and `SecondaryStatus` values change depending on the training scenario:

- **Spot instances acquired with no interruption during training**
  1. `InProgress`: Starting → Downloading → Training → Uploading
- **Spot instances interrupted once. Later, enough spot instances were acquired to finish the training job.**
  1. `InProgress`: Starting → Downloading → Training → Interrupted → Starting → Downloading → Training → Uploading
- **Spot instances interrupted twice and `MaxWaitTimeInSeconds` exceeded.**
  1. `InProgress`: Starting → Downloading → Training → Interrupted → Starting → Downloading → Training → Interrupted → Downloading → Training
  2. `Stopping`: Stopping
  3. `Stopped`: `MaxWaitTimeExceeded`
- **Spot instances were never launched.**
  1. `InProgress`: Starting
  2. `Stopping`: Stopping
  3. `Stopped`: `MaxWaitTimeExceeded`

## Use Checkpoints in Amazon SageMaker

Use checkpoints in Amazon SageMaker to save the state of machine learning (ML) models during training. Checkpoints are snapshots of the model and can be configured by the callback functions of ML frameworks. You can use the saved checkpoints to restart a training job from the last saved checkpoint.

The SageMaker training mechanism uses training containers on Amazon EC2 instances, and the checkpoint files are saved under a local directory of the containers. SageMaker provides functionality to copy the checkpoints from the local path to Amazon S3. Using checkpoints, you can do the following:

- Save your model snapshots under training due to an unexpected interruption to the training job or instance.
- Resume training the model in the future from a checkpoint.
- Analyze the model at intermediate stages of training.
- Use checkpoints with SageMaker managed spot training to save on training costs.

If you are using checkpoints with SageMaker managed spot training, SageMaker manages checkpointing your model training on a spot instance and resuming the training job on the next spot instance. With SageMaker managed spot training, you can significantly reduce the billable time for training ML models. For more information, see [Managed Spot Training in Amazon SageMaker \(p. 1859\)](#).

### Topics

- [Checkpoints for Frameworks and Algorithms in SageMaker \(p. 1861\)](#)
- [Enable Checkpointing \(p. 1862\)](#)
- [Browse Checkpoint Files \(p. 1863\)](#)
- [Resume Training From a Checkpoint \(p. 1863\)](#)
- [Considerations for Checkpointing \(p. 1864\)](#)

## Checkpoints for Frameworks and Algorithms in SageMaker

Use checkpoints to save snapshots of ML models built on your preferred frameworks within SageMaker.

### SageMaker frameworks and algorithms that support checkpointing

SageMaker supports checkpointing for Deep Learning Containers and a subset of built-in algorithms without requiring training script changes. SageMaker saves the checkpoints to the default local path '/opt/ml/checkpoints' and copies them to Amazon S3.

- Deep Learning Containers: [TensorFlow](#), [PyTorch](#), [MXNet](#), and [HuggingFace](#)

#### Note

If you are using the HuggingFace framework estimator, you need to specify a checkpoint output path through hyperparameters. For more information, see [Run training on Amazon SageMaker](#) in the *HuggingFace documentation*.

- Built-in algorithms: [Image Classification](#), [Object Detection](#), [Semantic Segmentation](#), and [XGBoost \(0.90-1 or later\)](#)

#### Note

If you are using the XGBoost algorithm in framework mode (script mode), you need to bring an XGBoost training script with checkpointing that's manually configured. For more information about the XGBoost training methods to save model snapshots, see [Training XGBoost](#) in the *XGBoost Python SDK documentation*.

If a pre-built algorithm that does not support checkpointing is used in a managed spot training job, SageMaker does not allow a maximum wait time greater than an hour for the job in order to limit wasted training time from interrupts.

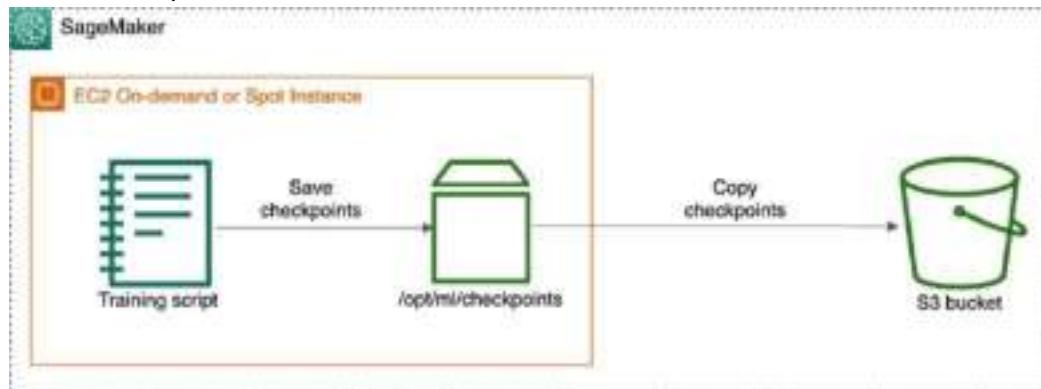
### For custom training containers and other frameworks

If you are using your own training containers, training scripts, or other frameworks not listed in the previous section, you must properly set up your training script using callbacks or training APIs to save

checkpoints to the local path ('/opt/ml/checkpoints') and load from the local path in your training script. SageMaker estimators can sync up with the local path and save the checkpoints to Amazon S3.

## Enable Checkpointing

After you enable checkpointing, SageMaker saves checkpoints to Amazon S3 and syncs your training job with the checkpoint S3 bucket.



The following example shows how to configure checkpoint paths when you construct a SageMaker estimator. To enable checkpointing, add the `checkpoint_s3_uri` and `checkpoint_local_path` parameters to your estimator.

The following example template shows how to create a generic SageMaker estimator and enable checkpointing. You can use this template for the supported algorithms by specifying the `image_uri` parameter. To find Docker image URIs for algorithms with checkpointing supported by SageMaker, see [Docker Registry Paths and Example Code \(p. 725\)](#). You can also replace `estimator` and `Estimator` with other SageMaker frameworks' estimator parent classes and estimator classes, such as [TensorFlow](#), [PyTorch](#), [MXNet](#), [HuggingFace](#) and [XGBoost](#).

```

import sagemaker
from sagemaker.estimator import Estimator

bucket=sagemaker.Session().default_bucket()
base_job_name="sagemaker-checkpoint-test"
checkpoint_in_bucket="checkpoints"

# The S3 URI to store the checkpoints
checkpoint_s3_bucket="s3://{}//{}//{}".format(bucket, base_job_name, checkpoint_in_bucket)

# The local path where the model will save its checkpoints in the training container
checkpoint_local_path="/opt/ml/checkpoints"

estimator = Estimator(
    ...
    image_uri="

```

The following two parameters specify paths for checkpointing:

- `checkpoint_local_path` – Specify the local path where the model saves the checkpoints periodically in a training container. The default path is set to '/opt/ml/checkpoints'. If you are

using other frameworks or bringing your own training container, ensure that your training script's checkpoint configuration specifies the path to '/opt/ml/checkpoints'.

**Note**

We recommend specifying the local paths as '/opt/ml/checkpoints' to be consistent with the default SageMaker checkpoint settings. If you prefer to specify your own local path, make sure you match the checkpoint saving path in your training script and the `checkpoint_local_path` parameter of the SageMaker estimators.

- `checkpoint_s3_uri` – The URI to an S3 bucket where the checkpoints are stored in real time.

To find a complete list of SageMaker estimator parameters, see the [Estimator API](#) in the [Amazon SageMaker Python SDK documentation](#).

## Browse Checkpoint Files

Locate checkpoint files using the SageMaker Python SDK and the Amazon S3 console.

### To find the checkpoint files programmatically

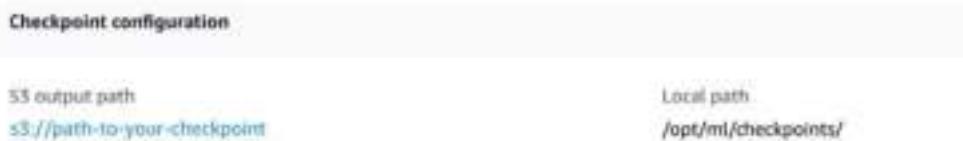
To retrieve the S3 bucket URI where the checkpoints are saved, check the following estimator attribute:

```
estimator.checkpoint_s3_uri
```

This returns the Amazon S3 output path for checkpoints configured while requesting the `CreateTrainingJob` request. To find the saved checkpoint files using the Amazon S3 console, use the following procedure.

### To find the checkpoint files from the Amazon S3 console

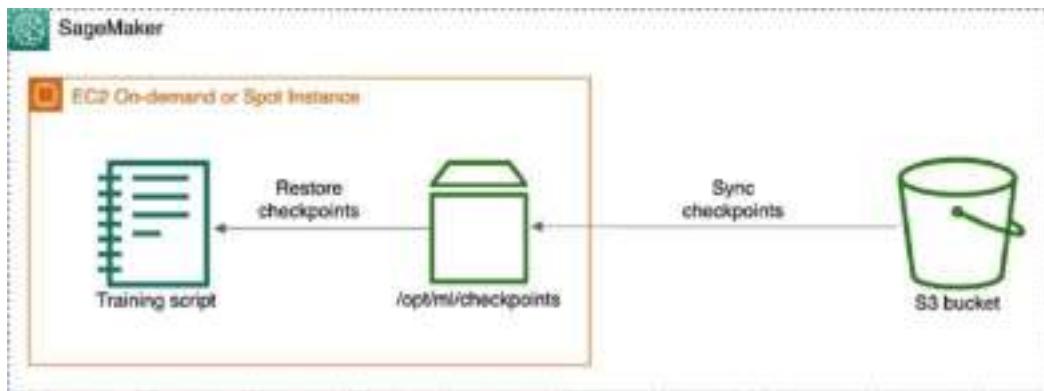
1. Sign in to the Amazon Web Services Management Console and open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the left navigation pane, choose **Training jobs**.
3. Choose the link to the training job with checkpointing enabled to open **Job settings**.
4. On the **Job settings** page of the training job, locate the **Checkpoint configuration** section.



5. Use the link to the S3 bucket to access the checkpoint files.

## Resume Training From a Checkpoint

To resume a training job from a checkpoint, run a new estimator with the same `checkpoint_s3_uri` that you created in the [Enable Checkpointing \(p. 1862\)](#) section. Once the training has resumed, the checkpoints from this S3 bucket are restored to `checkpoint_local_path` in each instance of the new training job. Ensure that the S3 bucket is in the same Region as that of the current SageMaker session.



## Considerations for Checkpointing

Consider the following when using checkpoints in SageMaker.

- To avoid overwrites in distributed training with multiple instances, you must manually configure the checkpoint file names and paths in your training script. The high-level SageMaker checkpoint configuration specifies a single Amazon S3 location without additional suffixes or prefixes to tag checkpoints from multiple instances.
- The SageMaker Python SDK does not support high-level configuration for checkpointing frequency. To control the checkpointing frequency, modify your training script using the framework's model save functions or checkpoint callbacks.
- If you use SageMaker checkpoints with SageMaker Debugger and SageMaker distributed and are facing issues, see the following pages for troubleshooting and considerations.
  - [Considerations for Amazon SageMaker Debugger \(p. 1741\)](#)
  - [Data Parallel Troubleshooting \(p. 1794\)](#)
  - [Model Parallel Troubleshooting \(p. 1817\)](#)

## Provide Dataset Metadata to Training Jobs with an Augmented Manifest File

To include metadata with your dataset in a training job, use an augmented manifest file. When using an augmented manifest file, your dataset must be stored in Amazon Simple Storage Service (Amazon S3) and you must configure your training job to use dataset stored there. You specify the location and format of this dataset for one or more [Channel](#). Augmented manifests can only support Pipe input mode. See the section, [InputMode](#) in [Channel](#) to learn more about pipe input mode.

When specifying a channel's parameters, you specify a path to the file, called a `S3Uri`. Amazon SageMaker interprets this URI based on the specified `S3DataType` in `S3DataSource`. The `AugmentedManifestFile` option defines a manifest format that includes metadata with the input data. Using an augmented manifest file is an alternative to preprocessing when you have labeled data. For training jobs using labeled data, you typically need to preprocess the dataset to combine input data with metadata before training. If your training dataset is large, preprocessing can be time consuming and expensive.

## Augmented Manifest File Format

An augmented manifest file must be formatted in [JSON Lines](#) format. In JSON Lines format, each line in the file is a complete JSON object followed by a newline separator.

During training, SageMaker parses each JSON line and sends some or all of its attributes on to the training algorithm. You specify which attribute contents to pass and the order in which to pass them with the `AttributeNames` parameter of the [CreateTrainingJob](#) API. The `AttributeNames` parameter is an ordered list of attribute names that SageMaker looks for in the JSON object to use as training input.

For example, if you list `["line", "book"]` for `AttributeNames`, the input data must include the attribute names of `line` and `book` in the specified order. For this example, the following augmented manifest file content is valid:

```
{"author": "Herman Melville", "line": "Call me Ishmael", "book": "Moby Dick"}  
{"line": "It was love at first sight.", "author": "Joseph Heller", "book": "Catch-22"}
```

SageMaker ignores unlisted attribute names even if they precede, follow, or are in between listed attributes.

When using augmented manifest files, observe the following guidelines:

- The order of the attributes listed in the `AttributeNames` parameter determines the order of the attributes passed to the algorithm in the training job.
- The listed `AttributeNames` can be a subset of all of the attributes in the JSON line. SageMaker ignores unlisted attributes in the file.
- You can specify any type of data allowed by the JSON format in `AttributeNames`, including text, numerical, data arrays, or objects.
- To include an S3 URI as an attribute name, add the suffix `-ref` to it.

If an attribute name contains the suffix `-ref`, the attribute's value must be an S3 URI to a data file that is accessible to the training job. For example, if `AttributeNames` contains `["image-ref", "is-a-cat"]`, a valid augmented manifest file might contain these lines:

```
{"image-ref": "s3://mybucket/sample01/image1.jpg", "is-a-cat": 1}  
{"image-ref": "s3://mybucket/sample02/image2.jpg", "is-a-cat": 0}
```

For the first line of this manifest, SageMaker retrieves the contents of the S3 object `s3://mybucket/foo/image1.jpg` and streams it to the algorithm for training. The second line is the string representation of the `is-a-cat` attribute `"1"`, which is followed by the contents of the second line.

## Stream Augmented Manifest File Data

Augmented manifest format enables you to do training in Pipe mode using files without needing to create RecordIO files. You need to specify both train and validation channels as values for the `InputDataConfig` parameter of the [CreateTrainingJob](#) request. Augmented manifest files are supported only for channels using Pipe input mode. For each channel, the data is extracted from its augmented manifest file and streamed (in order) to the algorithm through the channel's named pipe. Pipe mode uses the first in first out (FIFO) method, so records are processed in the order in which they are queued. For information about Pipe input mode, see [Input Mode](#).

Attribute names with a `"-ref"` suffix point to preformatted binary data. In some cases, the algorithm knows how to parse the data. In other cases, you might need to wrap the data so that records are delimited for the algorithm. If the algorithm is compatible with [RecordIO-formatted data](#), specifying `RecordIO` for `RecordWrapperType` solves this issue. If the algorithm is not compatible with `RecordIO` format, specify `None` for `RecordWrapperType` and make sure that your data is parsed correctly for your algorithm.

Using the `["image-ref", "is-a-cat"]` example, if you use `RecordIO` wrapping, the following stream of data is sent to the queue:

```
recordio_formatted(s3://mybucket/foo/  
image1.jpg)recordio_formatted("1")recordio_formatted(s3://mybucket/bar/  
image2.jpg)recordio_formatted("0")
```

Images that are not wrapped with RecordIO format, are streamed with the corresponding `is-a-cat` attribute value as one record. This can cause a problem because the algorithm might not delimit the images and attributes correctly. For more information about using augmented manifest files for image classification, see [Train with Augmented Manifest Image Format](#).

With augmented manifest files and Pipe mode in general, size limits of the EBS volume do not apply. This includes settings that otherwise must be within the EBS volume size limit such as `S3DataDistributionType`. For more information about Pipe mode and how to use it, see [Using Your Own Training Algorithms - Input Data Configuration](#).

## Use an Augmented Manifest File (Console)

To complete this procedure, you need:

- The URL of the S3 bucket where you've stored the augmented manifest file.
- To store the data that is listed in the augmented manifest file in an S3 bucket.
- The URL of the S3 bucket where you want to store the output of the job.

### To use an augmented manifest file in a training job (console)

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the navigation pane, choose **Training**, then choose **Training jobs**.
3. Choose **Create training job**.
4. Provide a name for the training job. The name must be unique within an Amazon Region in an Amazon account. It can have 1 to 63 characters. Valid characters: a-z, A-Z, 0-9, and . : + = @ \_ % - (hyphen).
5. Choose the algorithm that you want to use. For information about supported built-in algorithms, see [Use Amazon SageMaker Built-in Algorithms \(p. 718\)](#). If you want to use a custom algorithm, make sure that it is compatible with Pipe mode.
6. (Optional) For **Resource configuration**, either accept the default values or, to reduce computation time, increase the resource consumption.
  - a. (Optional) For **Instance type**, choose the ML compute instance type that you want to use. In most cases, **ml.m4.xlarge** is sufficient.
  - b. For **Instance count**, use the default, 1.
  - c. (Optional) For **Additional volume per instance (GB)**, choose the size of the ML storage volume that you want to provision. In most cases, you can use the default, 1. If you are using a large dataset, use a larger size.
7. Provide information about the input data for the training dataset.
  - a. For **Channel name**, either accept the default (**train**) or enter a more meaningful name, such as **training-augmented-manifest-file**.
  - b. For **InputMode**, choose **Pipe**.
  - c. For **S3 data distribution type**, choose **FullyReplicated**. When training incrementally, fully replicating causes each ML compute instance to use a complete copy of the expanded dataset. For neural-based algorithms, such as [Neural Topic Model \(NTM\) Algorithm \(p. 1457\)](#), choose **ShardedByS3Key**.
  - d. If the data specified in the augmented manifest file is uncompressed, set the **Compression type** to **None**. If the data is compressed using gzip, set it to **Gzip**.

- e. (Optional) For **Content type**, specify the appropriate MIME type. Content type is the multipurpose internet mail extension (MIME) type of the data.
- f. For **Record wrapper**, if the dataset specified in the augmented manifest file is saved in RecordIO format, choose **RecordIO**. If your dataset is not saved as a RecordIO-formatted file, choose **None**.
- g. For **S3 data type**, choose **AugmentedManifestFile**.
- h. For **S3 location**, provide the path to the bucket where you stored the augmented manifest file.
- i. For **AugmentedManifestFile attribute names**, specify the name of an attribute that you want to use. The attribute name must be present within the augmented manifest file, and is case-sensitive.
- j. (Optional) To add more attribute names, choose **Add row** and specify another attribute name for each attribute.
- k. (Optional) To adjust the order of attribute names, choose the up or down buttons next to the names. When using an augmented manifest file, the order of the specified attribute names is important.
- l. Choose **Done**.
8. For **Output data configuration**, provide the following information:
  - a. For **S3 location**, type the path to the S3 bucket where you want to store the output data.
  - b. (Optional) You can use your Amazon Key Management Service (Amazon KMS) encryption key to encrypt the output data at rest. For **Encryption key**, provide the key ID or its Amazon Resource Number (ARN). For more information, see [KMS-Managed Encryption Keys](#).
9. (Optional) For **Tags**, add one or more tags to the training job. A **tag** is metadata that you can define and assign to Amazon resources. In this case, you can use tags to help you manage your training jobs. A tag consists of a key and a value, which you define. For example, you might want to create a tag with **Project** as a key and a value that refers to a project that is related to the training job, such as **Home value forecasts**.
10. Choose **Create training job**. SageMaker creates and runs the training job.

After the training job has finished, SageMaker stores the model artifacts in the bucket whose path you provided for **S3 output path** in the **Output data configuration** field. To deploy the model to get predictions, see [Step 5: Deploy the Model to Amazon EC2 \(p. 63\)](#).

## Use an Augmented Manifest File (API)

The following shows how to train a model with an augmented manifest file using the SageMaker high-level Python library:

```
# Create a model object set to using "Pipe" mode.
model = sagemaker.estimator.Estimator(training_image,
                                       role,
                                       instance_count=1,
                                       instance_type='ml.p3.2xlarge',
                                       volume_size = 50,
                                       max_run = 360000,
                                       input_mode = 'Pipe',
                                       output_path=s3_output_location,
                                       sagemaker_session=session)

# Create a train data channel with S3_data_type as 'AugmentedManifestFile' and attribute
# names.
train_data = sagemaker.session.s3_input(your_augmented_manifest_file,
                                         distribution='FullyReplicated',
                                         content_type='application/x-recordio',
                                         s3_data_type='AugmentedManifestFile',
```

```
attribute_names=[ 'source-ref' , 'annotations' ],
input_mode='Pipe',
record_wrapping='RecordIO')

data_channels = {'train': train_data}

# Train a model.
model.fit(inputs=data_channels, logs=True)
```

After the training job has finished, SageMaker stores the model artifacts in the bucket whose path you provided for **S3 output path** in the **Output data configuration** field. To deploy the model to get predictions, see [Step 5: Deploy the Model to Amazon EC2 \(p. 63\)](#).

## Monitor and Analyze Training Jobs Using Metrics

An Amazon SageMaker training job is an iterative process that teaches a model to make predictions by presenting examples from a training dataset. Typically, a training algorithm computes several metrics, such as training error and prediction accuracy. These metrics help diagnose whether the model is learning well and will generalize well for making predictions on unseen data. The training algorithm writes the values of these metrics to logs, which SageMaker monitors and sends to Amazon CloudWatch in real time. To analyze the performance of your training job, you can view graphs of these metrics in CloudWatch. When a training job has completed, you can also get a list of the metric values that it computes in its final iteration by calling the [DescribeTrainingJob](#) operation.

### Topics

- [Training Metrics Sample Notebooks \(p. 1868\)](#)
- [Defining Training Metrics \(p. 1868\)](#)
- [Monitoring Training Job Metrics \( Console\) \(p. 1871\)](#)
- [Monitoring Training Job Metrics \(SageMaker Console\) \(p. 1871\)](#)
- [Example: Viewing a Training and Validation Curve \(p. 1873\)](#)

## Training Metrics Sample Notebooks

The following sample notebooks show how to view and plot training metrics:

- [An Introduction to the Amazon SageMaker ObjectToVec Model for Sequence-to-sequence Embedding \(object2vec\\_sentence\\_similarity.ipynb\)](#)
- [Regression with the Amazon SageMaker XGBoost Algorithm \(xgboost\\_abalone.ipynb\)](#)

For instructions how to create and access Jupyter notebook instances that you can use to run the examples in SageMaker, see [Example Notebooks \(p. 131\)](#). To see a list of all the SageMaker samples, after creating and opening a notebook instance, choose the **SageMaker Examples** tab. To access the example notebooks that show how to use training metrics, `object2vec_sentence_similarity.ipynb` and `xgboost_abalone.ipynb`, from the **Introduction to Amazon algorithms** section. To open a notebook, choose its **Use** tab, then choose **Create copy**.

## Defining Training Metrics

SageMaker automatically parses the logs for metrics that built-in algorithms emit and sends those metrics to CloudWatch. If you want SageMaker to parse logs from a custom algorithm and send metrics that the algorithm emits to CloudWatch, you have to specify the metrics that you want SageMaker to

send to CloudWatch when you configure the training job. You specify the name of the metrics that you want to send and the regular expressions that SageMaker uses to parse the logs that your algorithm emits to find those metrics.

You can specify the metrics that you want to track with the SageMaker console; the SageMaker Python SDK (<https://github.com/aws/sagemaker-python-sdk>), or the low-level SageMaker API.

### Topics

- [Defining Regular Expressions for Metrics \(p. 1869\)](#)
- [Defining Training Metrics \(Low-level SageMaker API\) \(p. 1869\)](#)
- [Defining Training Metrics \(SageMaker Python SDK\) \(p. 1870\)](#)
- [Define Training Metrics \(Console\) \(p. 1870\)](#)

## Defining Regular Expressions for Metrics

To find a metric, SageMaker searches the logs that your algorithm emits and finds logs that match the regular expression that you specify for that metric. If you are using your own algorithm, do the following:

- Make sure that the algorithm writes the metrics that you want to capture to logs
- Define a regular expression that accurately searches the logs to capture the values of the metrics that you want to send to CloudWatch metrics.

For example, suppose your algorithm emits metrics for training error and validation error by writing logs similar to the following to `stdout` or `stderr`:

```
Train_error=0.138318;  Valid_error = 0.324557;
```

If you want to monitor both of those metrics in CloudWatch, your `AlgorithmSpecification` would look like the following:

```
"AlgorithmSpecification": {  
    "TrainingImage": ContainerName,  
    "TrainingInputMode": "File",  
    "MetricDefinitions" : [  
        {  
            "Name": "train:error",  
            "Regex": "Train_error=(.*?);"  
        },  
        {  
            "Name": "validation:error",  
            "Regex": "Valid_error=(.*?);"  
        }  
    ]  
}
```

In the regex for the `train:error` metric defined above, the first part of the regex finds the exact text "Train\_error=", and the expression `(.*?)`; captures zero or more of any character until the first semicolon character. In this expression, the parenthesis tell the regex to capture what is inside them, . means any character, \* means zero or more, and ? means capture only until the first instance of the ; character.

## Defining Training Metrics (Low-level SageMaker API)

Define the metrics that you want to send to CloudWatch by specifying a list of metric names and regular expressions in the `MetricDefinitions` field of the `AlgorithmSpecification` input parameter

that you pass to the [CreateTrainingJob](#) operation. For example, if you want to monitor both the `train:error` and `validation:error` metrics in CloudWatch, your `AlgorithmSpecification` would look like the following:

```
"AlgorithmSpecification": {
    "TrainingImage": ContainerName,
    "TrainingInputMode": "File",
    "MetricDefinitions" : [
        {
            "Name": "train:error",
            "Regex": "Train_error=(.*?);"
        },
        {
            "Name": "validation:error",
            "Regex": "Valid_error=(.*?);"
        }
    ]
}
```

For more information about defining and running a training job by using the low-level SageMaker API, see [CreateTrainingJob](#).

## Defining Training Metrics (SageMaker Python SDK)

Define the metrics that you want to send to CloudWatch by specifying a list of metric names and regular expressions as the `metric_definitions` argument when you initialize an `Estimator` object. For example, if you want to monitor both the `train:error` and `validation:error` metrics in CloudWatch, your `Estimator` initialization would look like the following:

```
estimator =
    Estimator(image_name=ImageName,
              role='SageMakerRole',
              instance_count=1,
              instance_type='ml.c4.xlarge',
              k=10,
              sagemaker_session=sagemaker_session,
              metric_definitions=[
                  {'Name': 'train:error', 'Regex': 'Train_error=(.*?);'},
                  {'Name': 'validation:error', 'Regex': 'Valid_error=(.*?);'}
              ])

```

For more information about training by using [Amazon SageMaker Python SDK](#) estimators, see <https://github.com/aws/sagemaker-python-sdk#sagemaker-python-sdk-overview>.

## Define Training Metrics (Console)

You can define metrics for a custom algorithm in the console when you create a training job by providing the name and regular expression (regex) for **Metrics**.

For example, if you want to monitor both the `train:error` and `validation:error` metrics in CloudWatch, your metric definitions would look like the following:

```
[
    {
        "Name": "train:error",
        "Regex": "Train_error=(.*?);"
    },
    {
        "Name": "validation:error",
        "Regex": "Valid_error=(.*?);"
    }
]
```

```
{  
    "Name": "validation:error",  
    "Regex": "Valid_error=(.*?);"  
}  
]
```

## Monitoring Training Job Metrics ( Console)

You can monitor the metrics that a training job emits in real time in the CloudWatch console.

### To monitor training job metrics (CloudWatch console)

1. Open the CloudWatch console at <https://console.amazonaws.cn/cloudwatch>.
2. Choose **Metrics**, then choose [/aws/sagemaker/TrainingJobs](#).
3. Choose **TrainingJobName**.
4. On the **All metrics** tab, choose the names of the training metrics that you want to monitor.
5. On the **Graphed metrics** tab, configure the graph options. For more information about using CloudWatch graphs, see [Graph Metrics](#) in the *Amazon CloudWatch User Guide*.

## Monitoring Training Job Metrics (SageMaker Console)

You can monitor the metrics that a training job emits in real time by using the SageMaker console.

### To monitor training job metrics (SageMaker console)

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker>.
2. Choose **Training jobs**, then choose the training job whose metrics you want to see.
3. Choose **TrainingJobName**.
4. In the **Monitor** section, you can review the graphs of instance utilization and algorithm metrics.



## Example: Viewing a Training and Validation Curve

Typically, you split the data that you train your model on into training and validation datasets. You use the training set to train the model parameters that are used to make predictions on the training dataset. Then you test how well the model makes predictions by calculating predictions for the validation set. To analyze the performance of a training job, you commonly plot a training curve against a validation curve.

Viewing a graph that shows the accuracy for both the training and validation sets over time can help you to improve the performance of your model. For example, if training accuracy continues to increase over time, but, at some point, validation accuracy starts to decrease, you are likely overfitting your model. To address this, you can make adjustments to your model, such as increasing [regularization](#).

For this example, you can use the [Image-classification-full-training](#) example that is in the [Example notebooks](#) section of your SageMaker notebook instance. If you don't have a SageMaker notebook instance, create one by following the instructions at [Step 1: Create an Amazon SageMaker Notebook Instance \(p. 53\)](#). If you prefer, you can follow along with the [End-to-End Multiclass Image Classification Example](#) in the example notebook on GitHub. You also need an Amazon S3 bucket to store the training data and for the model output.

### To view training and validation error curves

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker>.
2. Choose **Notebooks**, and then choose **Notebook instances**.
3. Choose the notebook instance that you want to use, and then choose **Open**.
4. On the dashboard for your notebook instance, choose **SageMaker Examples**.
5. Expand the **Introduction to Amazon Algorithms** section, and then choose **Use** next to **Image-classification-fulltraining.ipynb**.
6. Choose **Create copy**. SageMaker creates an editable copy of the **Image-classification-fulltraining.ipynb** notebook in your notebook instance.
7. Run all of the cells in the notebook up to the **Inference** section. You don't need to deploy an endpoint or get inference for this example.
8. After the training job starts, open the CloudWatch console at <https://console.amazonaws.cn/cloudwatch>.
9. Choose **Metrics**, then choose [/aws/sagemaker/TrainingJobs](#).
10. Choose **TrainingJobName**.
11. On the **All metrics** tab, choose the **train:accuracy** and **validation:accuracy** metrics for the training job that you created in the notebook.
12. On the graph, choose an area that the metric's values to zoom in. You should see something like the following:

## Untitled graph



# Deploy Models for Inference

After you build and train your models, you can deploy them to get predictions in one of two ways:

- To set up a persistent endpoint to get predictions from your models, use Amazon SageMaker hosting services. For an overview on deploying a single model or multiple models with SageMaker hosting services, see [Deploy a Model on SageMaker Hosting Services \(p. 10\)](#).

For an example of how to deploy a model to the SageMaker hosting service, see [Deploy the Model to SageMaker Hosting Services \(p. 63\)](#).

Or, if you prefer, watch the following video tutorial:

[Deploy Your ML Models to Production at Scale with Amazon SageMaker](#)

- To get predictions for an entire dataset, use SageMaker batch transform. For an overview on deploying a model with SageMaker batch transform, see [Get Inferences for an Entire Dataset with Batch Transform \(p. 13\)](#).

For an example of how to deploy a model with batch transform, see [\(Optional\) Make Prediction with Batch Transform \(p. 64\)](#).

Or, if you prefer, watch the following video tutorial:

[Deploy Your ML Models to Production at Scale with Amazon SageMaker](#)

## Prerequisites

These topics assume that you have built and trained one or more machine learning models and are ready to deploy them. If you are new to SageMaker and have not completed these prerequisite tasks, work through the steps in the [Get Started with Amazon SageMaker \(p. 34\)](#) tutorial to familiarize yourself with an example of how SageMaker manages the data science process and how it handles model deployment. For more information about training a model, see [Train Models \(p. 713\)](#).

## What do you want to do?

SageMaker provides features to manage resources and optimize inference performance when deploying machine learning models. For guidance on using inference pipelines, compiling and deploying models with Neo, Elastic Inference, and automatic model scaling, see the following topics.

- To manage data processing and real-time predictions or to process batch transforms in a pipeline, see [Deploy an Inference Pipeline \(p. 1917\)](#).
- If you want to deploy a model on inf1 instances, see [Compile and Deploy Models with Neo \(p. 2021\)](#).
- To train TensorFlow, Apache MXNet, PyTorch, ONNX, and XGBoost models once and optimize them to deploy on ARM, Intel, and Nvidia processors, see [Compile and Deploy Models with Neo \(p. 2021\)](#).
- To preprocess entire datasets quickly or to get inferences from a trained model for large datasets when you don't need a persistent endpoint, see [Use Batch Transform \(p. 1888\)](#).
- To speed up the throughput and decrease the latency of getting real-time inferences from your deep learning models that are deployed as SageMaker hosted models using a GPU instance for your endpoint, see [Use Amazon SageMaker Elastic Inference \(EI\) \(p. 1876\)](#).

- To dynamically adjust the number of instances provisioned in response to changes in your workload, see [Automatically Scale Amazon SageMaker Models \(p. 1931\)](#).
- To create an endpoint that can host multiple models using a shared serving container, see [Host Multiple Models with Multi-Model Endpoints \(p. 1895\)](#).
- To test multiple models in production, see [Test models in production \(p. 1947\)](#).

## Manage Model Deployments

For guidance on managing model deployments, including monitoring, troubleshooting, and best practices, and for information on storage associated with inference hosting instances:

- For tools that can be used to monitor model deployments, see [Monitor Amazon SageMaker \(p. 2523\)](#).
- For troubleshooting model deployments, see [Troubleshoot Amazon SageMaker Model Deployments \(p. 1955\)](#).
- For model deployment best practices, see [Deployment Best Practices \(p. 1956\)](#).
- For information about the size of storage volumes provided for different sizes of hosting instances, see [Host Instance Storage Volumes \(p. 1946\)](#).

## Deploy Your Own Inference Code

For developers that need more advanced guidance on how to run your own inference code:

- To run your own inference code hosting services, see [Use Your Own Inference Code with Hosting Services \(p. 2167\)](#).
- To run your own inference code for batch transforms, see [Use Your Own Inference Code with Batch Transform \(p. 2172\)](#).

## Guide to SageMaker

[What Is Amazon SageMaker? \(p. 1\)](#)

## Use Amazon SageMaker Elastic Inference (EI)

This feature is not available in the China Regions.

By using Amazon Elastic Inference (EI), you can speed up the throughput and decrease the latency of getting real-time inferences from your deep learning models that are deployed as [Amazon SageMaker hosted models](#), but at a fraction of the cost of using a GPU instance for your endpoint. EI allows you to add inference acceleration to a hosted endpoint for a fraction of the cost of using a full GPU instance. Add an EI accelerator in one of the available sizes to a deployable model in addition to a CPU instance type, and then add that model as a production variant to an endpoint configuration that you use to deploy a hosted endpoint. You can also add an EI accelerator to a SageMaker [notebook instance](#) so that you can test and evaluate inference performance when you are building your models.

Elastic Inference is supported in EI-enabled versions of TensorFlow, Apache MXNet, and PyTorch. To use any other deep learning framework, export your model by using ONNX, and then import your model into

MXNet. You can then use your model with EI as an MXNet model. For information about importing an ONNX model into MXNet, see [Importing an ONNX model into MXNet](#).

#### Topics

- [How EI Works \(p. 1877\)](#)
- [Choose an EI Accelerator Type \(p. 1877\)](#)
- [Use EI in a SageMaker Notebook Instance \(p. 1878\)](#)
- [Use EI on a Hosted Endpoint \(p. 1878\)](#)
- [Frameworks that Support EI \(p. 1878\)](#)
- [Use EI with SageMaker Built-in Algorithms \(p. 1879\)](#)
- [EI Sample Notebooks \(p. 1879\)](#)
- [Set Up to Use EI \(p. 1879\)](#)
- [Attach EI to a Notebook Instance \(p. 1882\)](#)
- [Use EI on Amazon SageMaker Hosted Endpoints \(p. 1884\)](#)

## How EI Works

Amazon Elastic Inference accelerators are network attached devices that work along with SageMaker instances in your endpoint to accelerate your inference calls. Elastic Inference accelerates inference by allowing you to attach fractional GPUs to any SageMaker instance. You can select the client instance to run your application and attach an Elastic Inference accelerator to use the right amount of GPU acceleration for your inference needs. Elastic Inference helps you lower your cost when not fully utilizing your GPU instance for inference. We recommend trying Elastic Inference with your model using different CPU instances and accelerator sizes.

The following EI accelerator types are available. You can configure your endpoints or notebook instances with any EI accelerator type.

In the table, the throughput in teraflops (TFLOPS) is listed for both single-precision floating-point (F32) and half-precision floating-point (F16) operations. The memory in GB is also listed.

Accelerator Type	F32 Throughput in TFLOPS	F16 Throughput in TFLOPS	Memory in GB
ml.eia2.medium	1	8	2
ml.eia2.large	2	16	4
ml.eia2.xlarge	4	32	8
ml.eia1.medium	1	8	1
ml.eia1.large	2	16	2
ml.eia1.xlarge	4	32	4

## Choose an EI Accelerator Type

Consider the following factors when choosing an accelerator type for a hosted model:

- Models, input tensors and batch sizes influence the amount of accelerator memory you need. Start with an accelerator type that provides at least as much memory as the file size of your trained model. Factor in that a model might use significantly more memory than the file size at runtime.

- Demands on CPU compute resources, main system memory, and GPU-based acceleration and accelerator memory vary significantly between different kinds of deep learning models. The latency and throughput requirements of the application also determine the amount of compute and acceleration you need. Thoroughly test different configurations of instance types and EI accelerator sizes to make sure you choose the configuration that best fits the performance needs of your application.

For more information on selecting an EI accelerator, see:

- [Amazon Elastic Inference Overview](#)
- [Choosing an Instance and Accelerator Type for Your Model](#)
- [Optimizing costs in Amazon Elastic Inference with TensorFlow](#)

## Use EI in a SageMaker Notebook Instance

Typically, you build and test machine learning models in a SageMaker notebook before you deploy them for production. You can attach EI to your notebook instance when you create the notebook instance. You can set up an endpoint that is hosted locally on the notebook instance by using the local mode supported by TensorFlow, MXNet, and PyTorch estimators and models in the [Amazon SageMaker Python SDK](#) to test inference performance. Elastic Inference enabled PyTorch is not currently supported on notebook instances. For instructions on how to attach EI to a notebook instance and set up a local endpoint for inference, see [Attach EI to a Notebook Instance \(p. 1882\)](#). There are also Elastic Inference-enabled SageMaker Notebook Jupyter kernels for Elastic Inference-enabled versions of TensorFlow and Apache MXNet. For information about using SageMaker notebook instances, see [Use Amazon SageMaker Notebook Instances](#)

## Use EI on a Hosted Endpoint

When you are ready to deploy your model for production to provide inferences, you create a SageMaker hosted endpoint. You can attach EI to the instance where your endpoint is hosted to increase its performance at providing inferences. For instructions on how to attach EI to a hosted endpoint instance, see [Use EI on Amazon SageMaker Hosted Endpoints \(p. 1884\)](#).

## Frameworks that Support EI

Amazon Elastic Inference is designed to be used with Amazon enhanced versions of TensorFlow, Apache MXNet, or PyTorch machine learning frameworks. These enhanced versions of the frameworks are automatically built into containers when you use the Amazon SageMaker Python SDK, or you can download them as binary files and import them in your own Docker containers.

You can download the EI-enabled TensorFlow binary files from the public [amazonei-tensorflow](#) Amazon S3 bucket to the TensorFlow serving containers. For more information about building a container that uses the EI-enabled version of TensorFlow, see [Amazon Elastic Inference with TensorFlow in SageMaker](#).

You can download the EI-enabled MXNet binary files from the public [amazonei-apachemxnet](#) Amazon S3 bucket to the MXNet serving containers. For more information about building a container that uses the EI-enabled version of MXNet, see [Amazon Elastic Inference with MXNet in SageMaker](#).

You can download the EI-enabled PyTorch binary files from the public [amazonei-pytorch](#) Amazon S3 bucket to the PyTorch serving containers. For more information about building a container that uses the EI-enabled version of PyTorch, see [Amazon Elastic Inference with PyTorch in SageMaker](#).

To use Elastic Inference in a hosted endpoint, you can choose any of the following frameworks depending on your needs.

- [SageMaker Python SDK - Deploy TensorFlow models](#)
- [SageMaker Python SDK - Deploy MXNet models](#)
- [SageMaker Python SDK - Deploy PyTorch models](#)

If you need to create a custom container for deploying your model that is complex and requires extensions to a framework that the SageMaker pre-built containers do not support, use [the low-level Amazon SDK for Python \(Boto 3\)](#).

## Use EI with SageMaker Built-in Algorithms

Currently, the [Image Classification Algorithm \(p. 1399\)](#) and [Object Detection Algorithm \(p. 1479\)](#) built-in algorithms support EI. For an example that uses the Image Classification algorithm with EI, see [End-to-End Multiclass Image Classification Example](#).

## EI Sample Notebooks

The following Sample notebooks provide examples of using EI in SageMaker:

- [Using Amazon Elastic Inference with MXNet on Amazon SageMaker](#)
- [Using Amazon Elastic Inference with MXNet on an Amazon SageMaker Notebook Instance](#)
- [Using Amazon Elastic Inference with Neo-compiled TensorFlow model on SageMaker](#)
- [Using Amazon Elastic Inference with a pre-trained TensorFlow Serving model on SageMaker](#)

## Set Up to Use EI

Use the instructions in this topic only if one of the following applies to you:

- You want to use a customized role or permission policy.
- You want to use a VPC for your hosted model or notebook instance.

### Note

If you already have an execution role that has the `AmazonSageMakerFullAccess` managed policy attached (this is true for any IAM role that you create when you create a notebook instance, training job, or model in the console) and you are not connecting to an EI model or notebook instance in a VPC, you do not need to make any of these changes to use EI in Amazon SageMaker.

### Topics

- [Set Up Required Permissions \(p. 1879\)](#)
- [Use a Custom VPC to Connect to EI \(p. 1882\)](#)

## Set Up Required Permissions

To use EI in SageMaker, the role that you use to open a notebook instance or create a deployable model must have a policy with the required permissions attached. You can attach the `AmazonSageMakerFullAccess` managed policy, which contains the required permissions, to the role, or you can add a custom policy that has the required permissions. For information about creating an IAM role, see [Creating a Role for an Amazon Service \(Console\)](#) in the *Amazon Identity and Access Management User Guide*. For information about attaching a policy to a role, see [Adding and Removing IAM Policies](#).

Add these permissions specifically for connecting EI in an IAM policy.

```
{  
    "Effect": "Allow",  
    "Action": [  
        "elastic-inference:Connect",  
        "ec2:DescribeVpcEndpoints"  
    ],  
    "Resource": "*"  
}
```

The following IAM policy is the complete list of required permissions to use EI in SageMaker.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "elastic-inference:Connect",  
                "ec2:DescribeVpcEndpoints"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sagemaker:*"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ecr:GetAuthorizationToken",  
                "ecr:GetDownloadUrlForLayer",  
                "ecr:BatchGetImage",  
                "ecr:BatchCheckLayerAvailability",  
                "cloudwatch:PutMetricData",  
                "cloudwatch:PutMetricAlarm",  
                "cloudwatch:DescribeAlarms",  
                "cloudwatch:DeleteAlarms",  
                "ec2>CreateNetworkInterface",  
                "ec2:CreateNetworkInterfacePermission",  
                "ec2>DeleteNetworkInterface",  
                "ec2:DeleteNetworkInterfacePermission",  
                "ec2:DescribeNetworkInterfaces",  
                "ec2:DescribeVpcs",  
                "ec2:DescribeDhcpOptions",  
                "ec2:DescribeSubnets",  
                "ec2:DescribeSecurityGroups",  
                "application-autoscaling:DeleteScalingPolicy",  
                "application-autoscaling:DeleteScheduledAction",  
                "application-autoscaling:DeregisterScalableTarget",  
                "application-autoscaling:DescribeScalableTargets",  
                "application-autoscaling:DescribeScalingActivities",  
                "application-autoscaling:DescribeScalingPolicies",  
                "application-autoscaling:DescribeScheduledActions",  
                "application-autoscaling:PutScalingPolicy",  
                "application-autoscaling:PutScheduledAction",  
                "application-autoscaling:RegisterScalableTarget",  
                "logs>CreateLogGroup",  
                "logs>CreateLogStream",  
            ]  
        }  
    ]  
}
```

```

        "logs:DescribeLogStreams",
        "logs:GetLogEvents",
        "logs:PutLogEvents"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
    ],
    "Resource": [
        "arn:aws:s3::::*SageMaker*",
        "arn:aws:s3::::*Sagemaker*",
        "arn:aws:s3::::*sagemaker*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3>CreateBucket",
        "s3:GetBucketLocation",
        "s3>ListBucket",
        "s3>ListAllMyBuckets"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3GetObject"
    ],
    "Resource": "*",
    "Condition": {
        "StringEqualsIgnoreCase": {
            "s3:ExistingObjectTag/SageMaker": "true"
        }
    }
},
{
    "Action": "iam>CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::::role/aws-service-role/sagemaker.application-
autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "sagemaker.application-autoscaling.amazonaws.com"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "iam:PassRole"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "iam:PassedToService": "sagemaker.amazonaws.com"
        }
    }
}
]
}

```

## Use a Custom VPC to Connect to EI

To use EI with SageMaker in a VPC, you need to create and configure two security groups, and set up a PrivateLink VPC interface endpoint. EI uses VPC interface endpoint to communicate with SageMaker endpoints in your VPC. The security groups you create are used to connect to the VPC interface endpoint.

### Set up Security Groups to Connect to EI

To use EI within a VPC, you need to create two security groups:

- A security group to control access to the VPC interface endpoint that you will set up for EI.
- A security group that allows SageMaker to call into the first security group.

#### To configure the two security groups

1. Create a security group with no outbound connections. You will attach this to the VPC endpoint interface you create in the next section.
2. Create a second security group with no inbound connections, but with an outbound connection to the first security group.
3. Edit the first security group to allow inbound connections only to the second security group and all outbound connections.

For more information about VPC security groups, see [Security Groups for Your VPC](#) in the *Amazon Virtual Private Cloud User Guide*.

### Set up a VPC Interface Endpoint to Connect to EI

To use EI with SageMaker in a custom VPC, you need to set up a VPC interface endpoint (PrivateLink) for the EI service.

- Set up a VPC interface endpoint (PrivateLink) for the EI. Follow the instructions at [Creating an Interface Endpoint](#). In the list of services, choose **com.amazonaws.<region>.elastic-inference.runtime**. For **Security group**, make sure you select the first security group you created in the previous section to the endpoint.
- When you set up the interface endpoint, choose all of the Availability Zones where EI is available. EI fails if you do not set up at least two Availability Zones. For information about VPC subnets, see [VPCs and Subnets](#).

## Attach EI to a Notebook Instance

To test and evaluate inference performance using EI, you can attach EI to a notebook instance when you create or update a notebook instance. You can then use EI in local mode to host a model at an endpoint hosted on the notebook instance. You should test various sizes of notebook instances and EI accelerators to evaluate the configuration that works best for your use case.

### Set Up to Use EI

To use EI locally in a notebook instance, create a notebook instance with an EI instance.

#### To create a notebook instance with an EI instance

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.

2. In the navigation pane, choose **Notebook instances**.
3. Choose **Create notebook instance**.
4. For **Notebook instance name**, provide a unique name for your notebook instance.
5. For **notebook instance type**, choose a CPU instance such as ml.t2.medium.
6. For **Elastic Inference (EI)**, choose an instance from the list, such as **ml.eia2.medium**.
7. For **IAM role**, choose an IAM role that has the required permissions to use SageMaker and EI.
8. (Optional) For **VPC - Optional**, if you want the notebook instance to use a VPC, choose one from the available list. Otherwise, leave it as **No VPC**. If you use a VPC follow the instructions at [Use a Custom VPC to Connect to EI \(p. 1882\)](#).
9. (Optional) For **Lifecycle configuration - optional**, either leave it as **No configuration** or choose a lifecycle configuration. For more information, see [Customize a Notebook Instance Using a Lifecycle Configuration Script \(p. 124\)](#).
10. (Optional) For **Encryption key - optional**, if you want SageMaker to use an Amazon Key Management Service (Amazon KMS) key to encrypt data in the ML storage volume attached to the notebook instance, specify the key.
11. (Optional) For **Volume Size In GB - optional**, leave the default value of 5.
12. (Optional) For **Tags**, add tags to the notebook instance. A tag is a label you assign to help manage your notebook instances. A tag consists of a key and a value, both of which you define.
13. Choose **Create Notebook Instance**.

After you create your notebook instance with EI attached, you can create a Jupyter notebook and set up an EI endpoint that is hosted locally on the notebook instance.

#### Topics

- [Use EI in Local Mode in SageMaker \(p. 1883\)](#)

## Use EI in Local Mode in SageMaker

To use EI locally in an endpoint hosted on a notebook instance, use local mode with the [Amazon SageMaker Python SDK](#) versions of either the TensorFlow, MXNet, or PyTorch estimators or models. For more information about local mode support in the SageMaker Python SDK, see <https://github.com/aws/sagemaker-python-sdk#sagemaker-python-sdk-overview>.

#### Topics

- [Use EI in Local Mode with SageMaker TensorFlow Estimators and Models \(p. 1883\)](#)
- [Use EI in Local Mode with SageMaker Apache MXNet Estimators and Models \(p. 1884\)](#)
- [Use EI in Local Mode with SageMaker PyTorch Estimators and Models \(p. 1884\)](#)

## Use EI in Local Mode with SageMaker TensorFlow Estimators and Models

To use EI with TensorFlow in local mode, specify `local` for `instance_type` and `local_sagemaker_notebook` for `accelerator_type` when you call the `deploy` method of an estimator or a model object. For more information about [Amazon SageMaker Python SDK](#) TensorFlow estimators and models, see <https://sagemaker.readthedocs.io/en/stable/frameworks/tensorflow/index.html>.

The following code shows how to use local mode with an estimator object. To call the `deploy` method, you must have previously either:

- Trained the model by calling the `fit` method of an estimator.
- Pass a model artifact when you initialize the model object.

```
# Deploys the model to a local endpoint
tf_predictor = tf_model.deploy(initial_instance_count=1,
                               instance_type='local',
                               accelerator_type='local_sagemaker_notebook')
```

## Use EI in Local Mode with SageMaker Apache MXNet Estimators and Models

To use EI with MXNet in local mode, specify `local` for `instance_type` and `local_sagemaker_notebook` for `accelerator_type` when you call the `deploy` method of an estimator or a model object. For more information about [Amazon SageMaker Python SDK](https://sagemaker.readthedocs.io/en/stable/frameworks/mxnet/index.html) MXNet estimators and models, see <https://sagemaker.readthedocs.io/en/stable/frameworks/mxnet/index.html>.

The following code shows how to use local mode with an estimator object. You must have previously called the `fit` method of the estimator to train the model.

```
# Deploys the model to a local endpoint
mxnet_predictor = mxnet_estimator.deploy(initial_instance_count=1,
                                         instance_type='local',
                                         accelerator_type='local_sagemaker_notebook')
```

For a complete example of using EI in local mode with MXNet, see the sample notebook at [https://sagemaker-examples.readthedocs.io/en/latest/sagemaker-python-sdk/mxnet\\_mnist/mxnet\\_mnist\\_elastic\\_inference\\_local.html](https://sagemaker-examples.readthedocs.io/en/latest/sagemaker-python-sdk/mxnet_mnist/mxnet_mnist_elastic_inference_local.html).

## Use EI in Local Mode with SageMaker PyTorch Estimators and Models

To use EI with PyTorch in local mode, when you call the `deploy` method of an estimator or a model object, specify `local` for `instance_type` and `local_sagemaker_notebook` for `accelerator_type`. For more information about [Amazon SageMaker Python SDK](#) PyTorch estimators and models, see [SageMaker PyTorch Estimators and Models](#).

The following code shows how to use local mode with an estimator object. You must have previously called the `fit` method of the estimator to train the model.

```
# Deploys the model to a local endpoint
pytorch_predictor = pytorch_estimator.deploy(initial_instance_count=1,
                                              instance_type='local',
                                              accelerator_type='local_sagemaker_notebook')
```

## Use EI on Amazon SageMaker Hosted Endpoints

To use Elastic Inference (EI) in Amazon SageMaker with a hosted endpoint for real-time inference, specify an EI accelerator when you create the deployable model to be hosted at that endpoint. You can do this in one of the following ways:

- Use the [Amazon SageMaker Python SDK](#) versions of either the TensorFlow, MXNet, or PyTorch and the SageMaker pre-built containers for TensorFlow, MXNet, and PyTorch
- Build your own container, and use the low-level SageMaker API (Boto 3). You will need to import the EI-enabled version of either TensorFlow, MXNet, or PyTorch from the provided Amazon S3 locations into your container, and use one of those versions to write your training script.
- Use either the [Image Classification Algorithm \(p. 1399\)](#) or [Object Detection Algorithm \(p. 1479\)](#) build-in algorithms, and use the Amazon SDK for Python (Boto3) to run your training job and create your deployable model and hosted endpoint.

### Topics

- [Use EI with a SageMaker TensorFlow Container \(p. 1885\)](#)
- [Use EI with a SageMaker MXNet Container \(p. 1885\)](#)
- [Use EI with a SageMaker PyTorch Container \(p. 1886\)](#)
- [Use EI with Your Own Container \(p. 1886\)](#)

## Use EI with a SageMaker TensorFlow Container

To use TensorFlow with EI in SageMaker, you need to call the `deploy` method of either the [Estimator](#) or [Model](#) objects. You then specify an accelerator type using the `accelerator_type` input argument. For information on using TensorFlow in the SageMaker Python SDK, see: <https://sagemaker.readthedocs.io/en/stable/frameworks/tensorflow/index.html>.

SageMaker provides default model training and inference code for your convenience. For custom file formats, you might need to implement custom model training and inference code.

### Use an Estimator Object

To use an estimator object with EI, when you use the `deploy` method, include the `accelerator_type` input argument. The estimator returns a predictor object, which we call its `deploy` method, as shown in the example code.

```
# Deploy an estimator using EI (using the accelerator_type input argument)
predictor = estimator.deploy(initial_instance_count=1,
                             instance_type='ml.m4.xlarge',
                             accelerator_type='ml.eia2.medium')
```

### Use a Model Object

To use a model object with EI, when you use the `deploy` method, include the `accelerator_type` input argument. The estimator returns a predictor object, which we call its `deploy` method, as shown in the example code.

```
# Deploy a model using EI (using the accelerator_type input argument)
predictor = model.deploy(initial_instance_count=1,
                        instance_type='ml.m4.xlarge',
                        accelerator_type='ml.eia2.medium')
```

## Use EI with a SageMaker MXNet Container

To use MXNet with EI in SageMaker, you need to call the `deploy` method of either the [Estimator](#) or [Model](#) objects. You then specify an accelerator type using the `accelerator_type` input argument. For information about using MXNet in the [Amazon SageMaker Python SDK](#), see <https://sagemaker.readthedocs.io/en/stable/frameworks/mxnet/index.html>

For your convenience, SageMaker provides default model training and inference code. For custom file formats, you might need to write custom model training and inference code.

### Use an Estimator Object

To use an estimator object with EI, when you use the `deploy` method, include the `accelerator_type` input argument. The estimator returns a predictor object, which we call its `deploy` method, as shown in the example code.

```
# Deploy an estimator using EI (using the accelerator_type input argument)
```

```
predictor = estimator.deploy(initial_instance_count=1,  
                             instance_type='ml.m4.xlarge',  
                             accelerator_type='ml.eia2.medium')
```

## Use a Model Object

To use a model object with EI, when you use the `deploy` method, include the `accelerator_type` input argument. The estimator returns a predictor object, which we call its `deploy` method, as shown in the example code.

```
# Deploy a model using EI (using the accelerator_type input argument)  
predictor = model.deploy(initial_instance_count=1,  
                         instance_type='ml.m4.xlarge',  
                         accelerator_type='ml.eia2.medium')
```

For a complete example of using EI with MXNet in SageMaker, see the sample notebook at [https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/mxnet\\_mnist/mxnet\\_mnist\\_elastic\\_inference.ipynb](https://github.com/awslabs/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/mxnet_mnist/mxnet_mnist_elastic_inference.ipynb)

## Use EI with a SageMaker PyTorch Container

To use PyTorch with EI in SageMaker, you need to call the `deploy` method of either the [Estimator](#) or [Model](#) objects. You then specify an accelerator type using the `accelerator_type` input argument. For information about using PyTorch in the [Amazon SageMaker Python SDK](#), see [SageMaker PyTorch Estimators and Models](#).

For your convenience, SageMaker provides default model training and inference code. For custom file formats, you might need to write custom model training and inference code.

## Use an Estimator Object

To use an estimator object with EI, when you use the `deploy` method, include the `accelerator_type` input argument. The estimator returns a predictor object, which we call its `deploy` method, as shown in this example code.

```
# Deploy an estimator using EI (using the accelerator_type input argument)  
predictor = estimator.deploy(initial_instance_count=1,  
                             instance_type='ml.m4.xlarge',  
                             accelerator_type='ml.eia2.medium')
```

## Use a Model Object

To use a model object with EI, when you use the `deploy` method, include the `accelerator_type` input argument. The model returns a predictor object, which we call its `deploy` method, as shown in this example code.

```
# Deploy a model using EI (using the accelerator_type input argument)  
predictor = model.deploy(initial_instance_count=1,  
                         instance_type='ml.m4.xlarge',  
                         accelerator_type='ml.eia2.medium')
```

## Use EI with Your Own Container

To use EI with a model in a custom container that you build, use the low-level Amazon SDK for Python (Boto 3), download and import the Amazon EI-enabled versions of TensorFlow, Apache MXNet, or PyTorch machine learning frameworks, and write your training script using those frameworks.

## Import the EI Version of TensorFlow, MXNet, or PyTorch into Your Docker Container

To use EI with your own container, you need to import either the Amazon EI TensorFlow Serving library, the Amazon EI Apache MXNet library, or the Elastic Inference enabled PyTorch library into your container. The EI-enabled versions of TensorFlow and MXNet are currently available as binary files stored in Amazon S3 locations. You can download the EI-enabled binary for TensorFlow from the Amazon S3 bucket at [console.amazonaws.cn/s3/buckets/amazonei-tensorflow](https://console.amazonaws.cn/s3/buckets/amazonei-tensorflow). For information about building a container that uses the EI-enabled version of TensorFlow, see <https://github.com/aws/sagemaker-tensorflow-container#building-the-sagemaker-elastic-inference-tensorflow-serving-container>. You can download the EI-enabled binary for Apache MXNet from the public Amazon S3 bucket at [console.amazonaws.cn/s3/buckets/amazonei-apachemxnet](https://console.amazonaws.cn/s3/buckets/amazonei-apachemxnet). For information about building a container that uses the EI-enabled version of MXNet, see <https://github.com/aws/sagemaker-mxnet-container#building-the-sagemaker-elastic-inference-mxnet-container>. You can download the Elastic Inference enabled binary for PyTorch from the public Amazon S3 bucket at [console.amazonaws.cn/s3/buckets/amazonei-pytorch](https://console.amazonaws.cn/s3/buckets/amazonei-pytorch). For information about building a container that uses the Elastic Inference enabled version of PyTorch, see [Building your image](#).

## Create an EI Endpoint with Amazon SDK for Python (Boto 3)

To create an endpoint by using Amazon SDK for Python (Boto 3), you first create an endpoint configuration. The endpoint configuration specifies one or more models (called production variants) that you want to host at the endpoint. To attach EI to one or more of the production variants hosted at the endpoint, you specify one of the EI instance types as the `AcceleratorType` field for that `ProductionVariant`. You then pass that endpoint configuration when you create the endpoint.

### Create an Endpoint Configuration

To use EI, you need to specify an accelerator type in the endpoint configuration.

```
# Create Endpoint Configuration
from time import gmtime, strftime

endpoint_config_name = 'ImageClassificationEndpointConfig-' + strftime("%Y-%m-%d-%H-%M-%S",
    gmtime())
print(endpoint_config_name)
create_endpoint_config_response = sagemaker.create_endpoint_config(
    EndpointConfigName = endpoint_config_name,
    ProductionVariants=[{
        'InstanceType': 'ml.m4.xlarge',
        'InitialInstanceCount': 1,
        'ModelName': model_name,
        'VariantName': 'AllTraffic',
        'AcceleratorType': 'ml.eia2.medium'}])

print("Endpoint Config Arn: " + create_endpoint_config_response['EndpointConfigArn'])
```

### Create an Endpoint

After you create an endpoint configuration with an accelerator type, you can create an endpoint.

```
endpoint_name = 'ImageClassificationEndpoint-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
endpoint_response = sagemaker.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name)
```

After creating the endpoint, you can invoke it using the `invoke_endpoint` method in a Boto3 runtime object, as you would any other endpoint.

# Use Batch Transform

Use batch transform when you need to do the following:

- Preprocess datasets to remove noise or bias that interferes with training or inference from your dataset.
- Get inferences from large datasets.
- Run inference when you don't need a persistent endpoint.
- Associate input records with inferences to assist the interpretation of results.

To filter input data before performing inferences or to associate input records with inferences about those records, see [Associate Prediction Results with Input Records \(p. 1890\)](#). For example, you can filter input data to provide context for creating and interpreting reports about the output data.

For more information about batch transforms, see [Get Inferences for an Entire Dataset with Batch Transform \(p. 13\)](#).

## Topics

- [Use Batch Transform to Get Inferences from Large Datasets \(p. 1888\)](#)
- [Speed up a Batch Transform Job \(p. 1889\)](#)
- [Use Batch Transform to Test Production Variants \(p. 1889\)](#)
- [Batch Transform Errors \(p. 1890\)](#)
- [Batch Transform Sample Notebooks \(p. 1890\)](#)
- [Associate Prediction Results with Input Records \(p. 1890\)](#)

## Use Batch Transform to Get Inferences from Large Datasets

Batch transform automatically manages the processing of large datasets within the limits of specified parameters. For example, suppose that you have a dataset file, `input1.csv`, stored in an S3 bucket. The content of the input file might look like the following.:

```
Record1-Attribute1, Record1-Attribute2, Record1-Attribute3, ..., Record1-AttributeM  
Record2-Attribute1, Record2-Attribute2, Record2-Attribute3, ..., Record2-AttributeM  
Record3-Attribute1, Record3-Attribute2, Record3-Attribute3, ..., Record3-AttributeM  
...  
RecordN-Attribute1, RecordN-Attribute2, RecordN-Attribute3, ..., RecordN-AttributeM
```

When a batch transform job starts, SageMaker initializes compute instances and distributes the inference or preprocessing workload between them. Batch Transform partitions the Amazon S3 objects in the input by key and maps Amazon S3 objects to instances. When you have multiple files, one instance might process `input1.csv`, and another instance might process the file named `input2.csv`.

To keep large payloads below the `MaxPayloadInMB` limit, you can split an input file into several mini-batches. For example, you might create a mini-batch from `input1.csv` by including only two of the records.

```
Record3-Attribute1, Record3-Attribute2, Record3-Attribute3, ..., Record3-AttributeM  
Record4-Attribute1, Record4-Attribute2, Record4-Attribute3, ..., Record4-AttributeM
```

**Note**

SageMaker processes each input file separately. It doesn't combine mini-batches from different input files to comply with the [MaxPayloadInMB](#) limit.

To split input files into mini-batches, when you create a batch transform job, set the [SplitType](#) parameter value to `Line`. If `SplitType` is set to `None` or if an input file can't be split into mini-batches, SageMaker uses the entire input file in a single request.

If the batch transform job successfully processes all of the records in an input file, it creates an output file with the same name and the `.out` file extension. For multiple input files, such as `input1.csv` and `input2.csv`, the output files are named `input1.csv.out` and `input2.csv.out`. The batch transform job stores the output files in the specified location in Amazon S3, such as `s3://awsexamplebucket/output/`.

The predictions in an output file are listed in the same order as the corresponding records in the input file. The output file `input1.csv.out`, based on the input file shown earlier, would look like the following.

```
Inference1-Attribute1, Inference1-Attribute2, Inference1-Attribute3, ..., Inference1-AttributeM
Inference2-Attribute1, Inference2-Attribute2, Inference2-Attribute3, ..., Inference2-AttributeM
Inference3-Attribute1, Inference3-Attribute2, Inference3-Attribute3, ..., Inference3-AttributeM
...
InferenceN-Attribute1, InferenceN-Attribute2, InferenceN-Attribute3, ..., InferenceN-AttributeM
```

To combine the results of multiple output files into a single output file, set the [AssembleWith](#) parameter to `Line`.

When the input data is very large and is transmitted using HTTP chunked encoding, to stream the data to the algorithm, set [MaxPayloadInMB](#) to 0. Amazon SageMaker built-in algorithms don't support this feature.

For information about using the API to create a batch transform job, see the [CreateTransformJob](#) API. For more information about the correlation between batch transform input and output objects, see [OutputDataConfig](#). For an example of how to use batch transform, see [\(Optional\) Make Prediction with Batch Transform \(p. 64\)](#).

## Speed up a Batch Transform Job

If you are using the [CreateTransformJob](#) API, you can reduce the time it takes to complete batch transform jobs by using optimal values for parameters such as [MaxPayloadInMB](#), [MaxConcurrentTransforms](#), or [BatchStrategy](#). If you are using the SageMaker console, you can specify these optimal parameter values in the **Additional configuration** section of the **Batch transform job configuration** page. SageMaker automatically finds the optimal parameter settings for built-in algorithms. For custom algorithms, provide these values through an [execution-parameters](#) endpoint.

## Use Batch Transform to Test Production Variants

To test different models or various hyperparameter settings, create a separate transform job for each new model variant and use a validation dataset. For each transform job, specify a unique model name and location in Amazon S3 for the output file. To analyze the results, use [Inference Pipeline Logs and Metrics \(p. 1924\)](#).

## Batch Transform Errors

SageMaker uses the Amazon S3 [Multipart Upload API](#) to upload results from a batch transform job to Amazon S3. If an error occurs, the uploaded results are removed from Amazon S3. In some cases, such as when a network outage occurs, an incomplete multipart upload might remain in Amazon S3. To avoid incurring storage charges, we recommend that you add the [S3 bucket policy](#) to the S3 bucket lifecycle rules. This policy deletes incomplete multipart uploads that might be stored in the S3 bucket. For more information, see [Object Lifecycle Management](#).

If a batch transform job fails to process an input file because of a problem with the dataset, SageMaker marks the job as `Failed`. If an input file contains a bad record, the transform job doesn't create an output file for that input file because doing so prevents it from maintaining the same order in the transformed data as in the input file. When your dataset has multiple input files, a transform job continues to process input files even if it fails to process one. The processed files still generate useable results.

Exceeding the [MaxPayloadInMB](#) limit causes an error. This might happen with a large dataset if it can't be split, the [SplitType](#) parameter is set to `none`, or individual records within the dataset exceed the limit.

If you are using your own algorithms, you can use placeholder text, such as `ERROR`, when the algorithm finds a bad record in an input file. For example, if the last record in a dataset is bad, the algorithm places the placeholder text for that record in the output file.

## Batch Transform Sample Notebooks

For a sample notebook that uses batch transform with a principal component analysis (PCA) model to reduce data in a user-item review matrix, followed by the application of a density-based spatial clustering of applications with noise (DBSCAN) algorithm to cluster movies, see [Batch Transform with PCA and DBSCAN Movie Clusters](#). For instructions on creating and accessing Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). After creating and opening a notebook instance, choose the **SageMaker Examples** tab to see a list of all the SageMaker examples. The topic modeling example notebooks that use the NTM algorithms are located in the **Advanced functionality** section. To open a notebook, choose its **Use** tab, then choose **Create copy**.

## Associate Prediction Results with Input Records

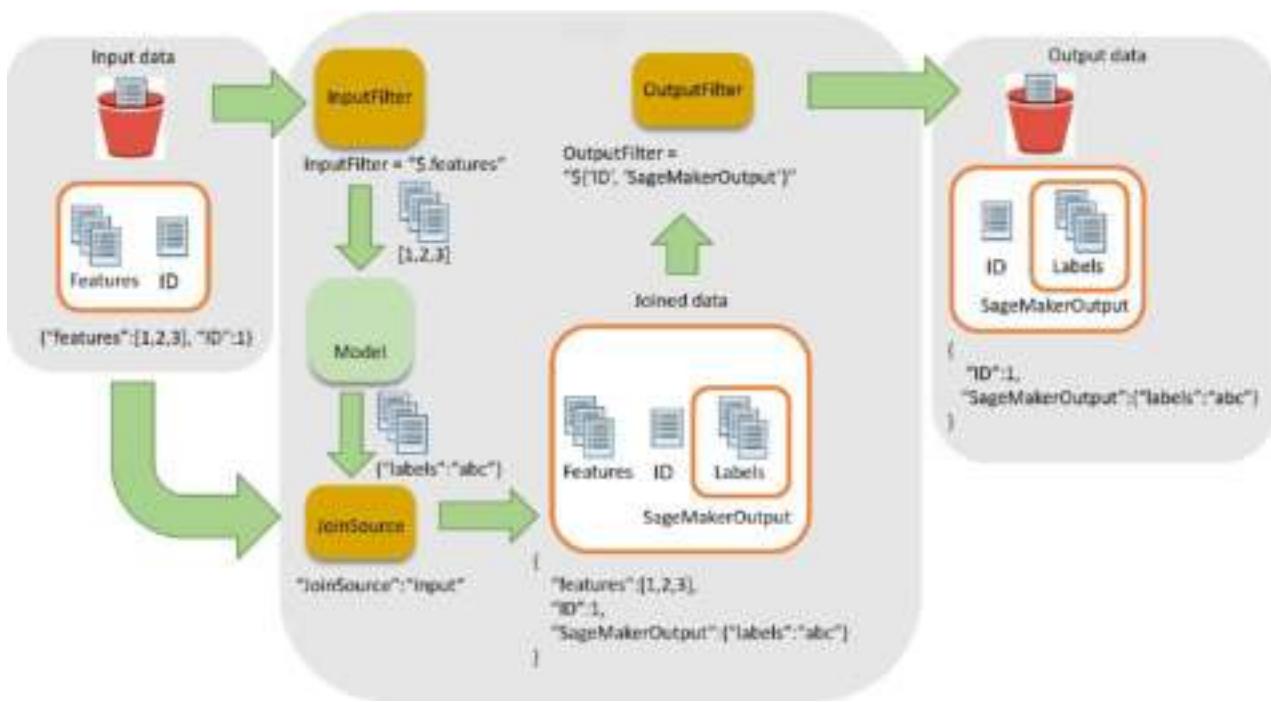
When making predictions on a large dataset, you can exclude attributes that aren't needed for prediction. After the predictions have been made, you can associate some of the excluded attributes with those predictions or with other input data in your report. By using batch transform to perform these data processing steps, you can often eliminate additional preprocessing or postprocessing. You can use input files in JSON and CSV format only.

### Topics

- [Workflow for Associating Inferences with Input Records \(p. 1890\)](#)
- [Use Data Processing in Batch Transform Jobs \(p. 1891\)](#)
- [Supported JSONPath Operators \(p. 1892\)](#)
- [Batch Transform Examples \(p. 1892\)](#)

## Workflow for Associating Inferences with Input Records

The following diagram shows the workflow for associating inferences with input records.



To associate inferences with input data, there are three main steps:

1. Filter the input data that is not needed for inference before passing the input data to the batch transform job. Use the [InputFilter](#) parameter to determine which attributes to use as input for the model.
2. Associate the input data with the inference results. Use the [JoinSource](#) parameter to combine the input data with the inference.
3. Filter the joined data to retain the inputs that are needed to provide context for interpreting the predictions in the reports. Use [OutputFilter](#) to store the specified portion of the joined dataset in the output file.

## Use Data Processing in Batch Transform Jobs

When creating a batch transform job with [CreateTransformJob](#) to process data:

1. Specify the portion of the input to pass to the model with the [InputFilter](#) parameter in the [DataProcessing](#) data structure.
2. Join the raw input data with the transformed data with the [JoinSource](#) parameter.
3. Specify which portion of the joined input and transformed data from the batch transform job to include in the output file with the [OutputFilter](#) parameter.
4. Choose either JSON- or CSV-formatted files for input:
  - For JSON- or JSON Lines-formatted input files, SageMaker either adds the [SageMakerOutput](#) attribute to the input file or creates a new JSON output file with the [SageMakerInput](#) and [SageMakerOutput](#) attributes. For more information, see [DataProcessing](#).
  - For CSV-formatted input files, the joined input data is followed by the transformed data and the output is a CSV file.

If you use an algorithm with the [DataProcessing](#) structure, it must support your chosen format for *both* input and output files. For example, with the [TransformOutput](#) field of the

CreateTransformJob API, you must set both the `Content Type` and `Accept` parameters to one of the following values: `text/csv`, `application/json`, or `application/jsonlines`. The syntax for specifying columns in a CSV file and specifying attributes in a JSON file are different. Using the wrong syntax causes an error. For more information, see [Batch Transform Examples \(p. 1892\)](#). For more information about input and output file formats for built-in algorithms, see [Use Amazon SageMaker Built-in Algorithms \(p. 718\)](#).

The record delimiters for the input and output must also be consistent with your chosen file input. The `SplitType` parameter indicates how to split the records in the input dataset. The `AssembleWith` parameter indicates how to reassemble the records for the output. If you set input and output formats to `text/csv`, you must also set the `SplitType` and `AssemblyType` parameters to `line`. If you set the input and output formats to `application/jsonlines`, you can set both `SplitType` and `AssemblyType` to `line`.

For JSON files, the attribute name `SageMakerOutput` is reserved for output. The JSON input file can't have an attribute with this name. If it does, the data in the input file might be overwritten.

## Supported JSONPath Operators

To filter and join the input data and inference, use a JSONPath subexpression. SageMaker supports only a subset of the defined JSONPath operators. The following table lists the supported JSONPath operators. For CSV data, each row is taken as a JSON array, so only index based JSONPaths can be applied, e.g. `[$[0], $[1 : ]]`. CSV data should also follow [RFC format](#).

JSONPath Operator	Description	Example
<code>\$</code>	The root element to a query. This operator is required at the beginning of all path expressions.	<code>\$</code>
<code>.&lt;name&gt;</code>	A dot-notated child element.	<code>\$.id</code>
<code>*</code>	A wildcard. Use in place of an attribute name or numeric value.	<code>\$.id.*</code>
<code>['&lt;name&gt;' (, '&lt;name&gt;')]</code>	A bracket-notated element or multiple child elements.	<code>\$['id', 'SageMakerOutput']</code>
<code>[&lt;number&gt; (, &lt;number&gt;)]</code>	An index or array of indexes. Negative index values are also supported. A <code>-1</code> index refers to the last element in an array.	<code>\$[1], \$[1, 3, 5]</code>
<code>[&lt;start&gt;:&lt;end&gt;]</code>	An array slice operator. The <code>array slice()</code> method extracts a section of an array and returns a new array. If you omit <code>&lt;start&gt;</code> , SageMaker uses the first element of the array. If you omit <code>&lt;end&gt;</code> , SageMaker uses the last element of the array.	<code>\$[2:5], \$[:5], \$[2:]</code>

When using the bracket-notation to specify multiple child elements of a given field, additional nesting of children within brackets is not supported. For example, `$.field1.[ 'child1', 'child2' ]` is supported while `$.field1.[ 'child1', 'child2.grandchild' ]` is not.

For more information about JSONPath operators, see [JsonPath](#) on GitHub.

## Batch Transform Examples

The following examples show some common ways to join input data with prediction results.

## Topics

- [Example: Output Only Inferences \(p. 1893\)](#)
- [Example: Output Input Data and Inferences \(p. 1893\)](#)
- [Example: Output an ID Column with Results and Exclude the ID Column from the Input \(CSV\) \(p. 1894\)](#)
- [Example: Output an ID Attribute with Results and Exclude the ID Attribute from the Input \(JSON\) \(p. 1894\)](#)

## Example: Output Only Inferences

By default, the `DataProcessing` parameter doesn't join inference results with input. It outputs only the inference results.

If you want to explicitly specify to not join results with input, use the [Amazon SageMaker Python SDK](#) and specify the following settings in a transformer call.

```
sm_transformer = sagemaker.transformer.Transformer(...)  
sm_transformer.transform(..., input_filter="$", join_source= "None", output_filter="$")
```

To output inferences using the Amazon SDK for Python, add the following code to your `CreateTransformJob` request. The following code mimics the default behavior.

```
{  
    "DataProcessing": {  
        "InputFilter": "$",  
        "JoinSource": "None",  
        "OutputFilter": "$"  
    }  
}
```

## Example: Output Input Data and Inferences

If you're using the [Amazon SageMaker Python SDK](#), to combine the input data with the inferences in the output file, specify "Input" for the `JoinSource` parameter in a transformer call.

```
sm_transformer = sagemaker.transformer.Transformer(...)  
sm_transformer.transform(..., join_source= "Input")
```

If you're using the Amazon SDK for Python (Boto 3), join all input data with the inference by adding the following code to your `CreateTransformJob` request.

```
{  
    "DataProcessing": {  
        "JoinSource": "Input"  
    }  
}
```

For JSON or JSON Lines input files, the results are in the `SageMakerOutput` key in the input JSON file. For example, if the input is a JSON file that contains the key-value pair `{"key":1}`, the data transform result might be `{"label":1}`.

SageMaker stores both in the input file in the `SageMakerInput` key.

```
{
  "key":1,
  "SageMakerOutput":{"label":1}
}
```

#### Note

The joined result for JSON must be a key-value pair object. If the input isn't a key-value pair object, SageMaker creates a new JSON file. In the new JSON file, the input data is stored in the `SageMakerInput` key and the results are stored as the `SageMakerOutput` value.

For a CSV file, for example, if the record is [1, 2, 3], and the label result is [1], then the output file would contain [1, 2, 3, 1].

### Example: Output an ID Column with Results and Exclude the ID Column from the Input (CSV)

If you are using the [Amazon SageMaker Python SDK](#), to include results or an ID column in the output, specify indexes of the joined dataset in a transformer call. For example, if your data includes five columns and the first one is the ID column, use the following transformer request.

```
sm_transformer = sagemaker.transformer.Transformer(...)
sm_transformer.transform(..., input_filter="[$[1:]", join_source= "Input", output_filter="$")
```

If you are using the Amazon SDK for Python (Boto 3), add the following code to your [CreateTransformJob](#) request.

```
{
  "DataProcessing": {
    "InputFilter": "$[1:]",
    "JoinSource": "Input",
    "OutputFilter": "$"
  }
}
```

To specify columns in SageMaker, use the index of the array elements. The first column is index 0, the second column is index 1, and the sixth column is index 5.

To exclude the first column from the input, set `InputFilter` to `"[$[1:]"`. The colon (`:`) tells SageMaker to include all of the elements between two values, inclusive. For example, `[$[1:4]` specifies the second through fifth columns.

If you omit the number after the colon, for example, `[5 : ]`, the subset includes all columns from the 6th column through the last column. If you omit the number before the colon, for example, `[:5]`, the subset includes all columns from the first column (index 0) through the sixth column.

### Example: Output an ID Attribute with Results and Exclude the ID Attribute from the Input (JSON)

If you are using the [Amazon SageMaker Python SDK](#), include results of an ID attribute in the output by specifying it in a transformer call. For example, if you store data in the `features` attribute and the record ID in the `ID` attribute, you would use the following transformer request.

```
sm_transformer = sagemaker.transformer.Transformer(...)
sm_transformer.transform(..., input_filter=".features", join_source= "Input",
                      output_filter="['id', 'SageMakerOutput']")
```

If you are using the Amazon SDK for Python (Boto 3), join all input data with the inference by adding the following code to your [CreateTransformJob](#) request.

```
{  
    "DataProcessing": {  
        "InputFilter": "$.features",  
        "JoinSource": "Input",  
        "OutputFilter": "$['id', 'SageMakerOutput']"  
    }  
}
```

**Warning**

If you are using a JSON-formatted input file, the file can't contain the attribute name `SageMakerOutput`. This attribute name is reserved for the output file. If your JSON-formatted input file contains an attribute with this name, values in the input file might be overwritten with the inference.

## Host Multiple Models with Multi-Model Endpoints

To create an endpoint that can host multiple models, use multi-model endpoints. Multi-model endpoints provide a scalable and cost-effective solution to deploying large numbers of models. They use a shared serving container that is enabled to host multiple models. This reduces hosting costs by improving endpoint utilization compared with using single-model endpoints. It also reduces deployment overhead because Amazon SageMaker manages loading models in memory and scaling them based on the traffic patterns to them.

Multi-model endpoints also enable time-sharing of memory resources across your models. This works best when the models are fairly similar in size and invocation latency. When this is the case, multi-model endpoints can effectively use instances across all models. If you have models that have significantly higher transactions per second (TPS) or latency requirements, we recommend hosting them on dedicated endpoints. Multi-model endpoints are also well suited to scenarios that can tolerate occasional cold-start-related latency penalties that occur when invoking infrequently used models.

Multi-model endpoints support A/B testing. They work with Auto Scaling and Amazon PrivateLink. You can use multi-model-enabled containers with serial inference pipelines, but only one multi-model-enabled container can be included in an inference pipeline. You can't use multi-model-enabled containers with Amazon Elastic Inference.

You can use the Amazon SDK for Python (Boto) or the SageMaker console to create a multi-model endpoint. You can use multi-model endpoints with custom-built containers by integrating the [Multi Model Server](#) library.

### Topics

- [Supported Algorithms and Frameworks \(p. 1896\)](#)
- [Sample Notebooks for Multi-Model Endpoints \(p. 1896\)](#)
- [How Multi-Model Endpoints Work \(p. 1896\)](#)
- [Setting SageMaker Multi-Model Endpoint Model Caching Behavior \(p. 1897\)](#)
- [Instance Recommendations for Multi-Model Endpoint Deployments \(p. 1897\)](#)
- [Create a Multi-Model Endpoint \(p. 1898\)](#)
- [Invoke a Multi-Model Endpoint \(p. 1902\)](#)
- [Add or Remove Models \(p. 1903\)](#)
- [Build Your Own Container with Multi Model Server \(p. 1904\)](#)
- [Multi-Model Endpoint Security \(p. 1909\)](#)

- [CloudWatch Metrics for Multi-Model Endpoint Deployments \(p. 1909\)](#)

## Supported Algorithms and Frameworks

The inference containers for the following algorithms and frameworks support multi-model endpoints:

- [XGBoost Algorithm \(p. 1524\)](#)
- [K-Nearest Neighbors \(k-NN\) Algorithm \(p. 1428\)](#)
- [Linear Learner Algorithm \(p. 1442\)](#)
- [Random Cut Forest \(RCF\) Algorithm \(p. 1494\)](#)
- [Use TensorFlow with Amazon SageMaker \(p. 31\)](#)
- [Use Scikit-learn with Amazon SageMaker \(p. 30\)](#)
- [Use Apache MXNet with Amazon SageMaker \(p. 16\)](#)
- [Use PyTorch with Amazon SageMaker \(p. 27\)](#)

To use any other framework or algorithm, use the SageMaker inference toolkit to build a container that supports multi-model endpoints. For information, see [Build Your Own Container with Multi Model Server \(p. 1904\)](#).

## Sample Notebooks for Multi-Model Endpoints

For a sample notebook that uses SageMaker to deploy multiple XGBoost models to an endpoint, see the [Multi-Model Endpoint XGBoost Sample Notebook](#). For a sample notebook that shows how to set up and deploy a custom container that supports multi-model endpoints in SageMaker, see the [Multi-Model Endpoint BYOC Sample Notebook](#). For instructions how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). After you've created a notebook instance and opened it, choose the **SageMaker Examples** tab to see a list of all the SageMaker samples. The Multi-Model Endpoint notebook is located in the **ADVANCED FUNCTIONALITY** section. To open a notebook, choose its **Use** tab and choose **Create copy**.

## How Multi-Model Endpoints Work

SageMaker manages the lifecycle of models hosted on multi-model endpoints in the container's memory. Instead of downloading all of the models from an Amazon S3 bucket to the container when you create the endpoint, SageMaker dynamically loads them when you invoke them. When SageMaker receives an invocation request for a particular model, it does the following:

1. Routes the request to an instance behind the endpoint.
2. Downloads the model from the S3 bucket to that instance's storage volume.
3. Loads the model to the container's memory on that instance. If the model is already loaded in the container's memory, invocation is faster because SageMaker doesn't need to download and load it.

SageMaker continues to route requests for a model to the instance where the model is already loaded. However, if the model receives many invocation requests, and there are additional instances for the multi-model endpoint, SageMaker routes some requests to another instance to accommodate the traffic. If the model isn't already loaded on the second instance, the model is downloaded to that instance's storage volume and loaded into the container's memory.

When an instance's memory utilization is high and SageMaker needs to load another model into memory, it unloads unused models from that instance's container to ensure that there is enough

memory to load the model. Models that are unloaded remain on the instance's storage volume and can be loaded into the container's memory later without being downloaded again from the S3 bucket. If the instance's storage volume reaches its capacity, SageMaker deletes any unused models from the storage volume.

To delete a model, stop sending requests and delete it from the S3 bucket. SageMaker provides multi-model endpoint capability in a serving container. Adding models to, and deleting them from, a multi-model endpoint doesn't require updating the endpoint itself. To add a model, you upload it to the S3 bucket and invoke it. You don't need code changes to use it.

When you update a multi-model endpoint, invocation requests on the endpoint might experience higher latencies as traffic is directed to the instances in the updated endpoint.

## Setting SageMaker Multi-Model Endpoint Model Caching Behavior

By default, multi-model endpoints cache frequently used models in memory and on disk to provide low latency inference. The cached models are unloaded and/or deleted from disk only when a container runs out of memory or disk space to accommodate a newly targeted model.

You can change the caching behavior of a multi-model endpoint and explicitly enable or disable model caching by setting the parameter `ModelCacheSetting` when you call `create_model`.

We recommend setting the value of the `ModelCacheSetting` parameter to `Disabled` for use cases that do not benefit from model caching. For example, when a large number of models need to be served from the endpoint but each model is invoked only once (or very infrequently). For such use cases, setting the value of the `ModelCacheSetting` parameter to `Disabled` allows higher transactions per second (TPS) for `invoke_endpoint` requests compared to the default caching mode. Higher TPS in these use cases is because SageMaker does the following after the `invoke_endpoint` request:

- Asynchronously unloads the model from memory and deletes it from disk immediately after it is invoked.
- Provides higher concurrency for downloading and loading models in the inference container. The concurrency is a factor of the number of vCPUs of the container instance.

For guidelines on choosing a SageMaker ML instance type for a multi-model endpoint, see [Instance Recommendations for Multi-Model Endpoint Deployments \(p. 1897\)](#).

## Instance Recommendations for Multi-Model Endpoint Deployments

There are several items to consider when selecting a SageMaker ML instance type for a multi-model endpoint. Provision sufficient [Amazon Elastic Block Store \(Amazon EBS\)](#) capacity for all of the models that need to be served. Balance performance (minimize cold starts) and cost (don't over-provision instance capacity). For information about the size of the storage volume that SageMaker attaches for each instance type for an endpoint and for a multi-model endpoint, see [Host Instance Storage Volumes \(p. 1946\)](#). For a container configured to run in `MultiModel` mode, the storage volume provisioned for its instances has more memory. This allows more models to be cached on the instance storage volume.

When choosing a SageMaker ML instance type, consider the following:

- Multi-model endpoints are not supported on GPU instance types.

- The traffic distribution (access patterns) to the models that you want to host behind the multi-model endpoint, along with the model size (how many models could be loaded in memory on the instance):
  - Think of the amount of memory on an instance as the cache space for models to be loaded. Think of the number of vCPUs as the concurrency limit to perform inference on the loaded models (assuming that invoking a model is bound to CPU).
  - A higher amount of instance memory enables you to have more models loaded and ready to serve inference requests. You don't need to waste time loading the model.
  - A higher amount of vCPUs enables you to invoke more unique models concurrently (again assuming that inference is bound to CPU).
  - Have some "slack" memory available so that unused models can be unloaded, and especially for multi-model endpoints with multiple instances. If an instance or an Availability Zone fails, the models on those instances will be rerouted to other instances behind the endpoint.
- Tolerance to loading/downloading times:
  - d instance type families (for example, m5d, c5d, or r5d) come with an NVMe (non-volatile memory express) SSD, which offers high I/O performance and might reduce the time it takes to download models to the storage volume and for the container to load the model from the storage volume.
  - Because d instance types come with an NVMe SSD storage, SageMaker does not attach an Amazon EBS storage volume to these ML compute instances that hosts the multi-model endpoint. Auto scaling works best when the models are similarly sized and homogenous, that is when they have similar inference latency and resource requirements.

In some cases, you might opt to reduce costs by choosing an instance type that can't hold all of the targeted models in memory at once. SageMaker dynamically unloads models when it runs out of memory to make room for a newly targeted model. For infrequently requested models, you are going to pay a price with the dynamic load latency. In cases with more stringent latency needs, you might opt for larger instance types or more instances. Investing time up front for proper performance testing and analysis will pay great dividends in successful production deployments.

You can use the `Average` statistic of the `ModelCacheHit` metric to monitor the ratio of requests where the model is already loaded. You can use the `SampleCount` statistic for the `ModelUnloadingTime` metric to monitor the number of unload requests sent to the container during a time period. If models are unloaded too frequently (an indicator of thrashing, where models are being unloaded and loaded again because there is insufficient cache space for the working set of models), consider using a larger instance type with more memory or increasing the number of instances behind the multi-model endpoint. For multi-model endpoints with multiple instances, be aware that a model might be loaded on more than 1 instance.

SageMaker multi-model endpoints fully supports Auto Scaling, which manages replicas of models to ensure models scale based on traffic patterns. We recommend that you configure your multi-model endpoint and the size of your instances by considering all of the above and also set up auto scaling for your endpoint. The invocation rates used to trigger an auto-scale event is based on the aggregate set of predictions across the full set of models served by the endpoint.

## Create a Multi-Model Endpoint

You can use Amazon SDK for Python (Boto) or Amazon SageMaker to create a multi-model endpoint.

### Topics

- [Create a Multi-Model Endpoint \(Console\) \(p. 1898\)](#)
- [Create a Multi-Model Endpoint \(Amazon SDK for Python \(Boto\)\) \(p. 1901\)](#)

## Create a Multi-Model Endpoint (Console)

**To create a multi-model endpoint (console)**

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Choose **Model**, and then from the **Inference** group, choose **Create model**.
3. For **Model name**, enter a name.
4. For **IAM role**, choose or create an IAM role that has the AmazonSageMakerFullAccess IAM policy attached.
5. In the **Container definition** section, for **Provide model artifacts and inference image options** choose **Use multiple models**.

The screenshot shows the 'Create model' page in the Amazon SageMaker console. At the top, there's a navigation bar with the Amazon logo, 'Services' dropdown, and 'Resource Groups'. Below that, a breadcrumb trail shows 'Amazon SageMaker > Models > Create model'. The main title 'Create model' is displayed prominently. A sub-section title 'Model settings' is visible. Under 'Model name', the value 'mml-test-model' is entered, with a note below stating 'Maximum of 63 alphanumeric characters. Can include your account in an Region.' Under 'IAM role', it says 'Amazon SageMaker requires permissions to call other services. Create a role that has the [AmazonSageMakerFullAccess](#) policy.' A specific role name 'AmazonSageMaker-ExecutionRole- XXXXX' is listed. At the bottom, there's a section titled 'Container definition 1'.

Services ▾ Resource Groups

Amazon SageMaker > Models > Create model

# Create model

To deploy a model to Amazon SageMaker, first upload inference code. See [Deploying a Model on Amazon SageMaker](#).

## Model settings

Model name

mml-test-model

Maximum of 63 alphanumeric characters. Can include your account in an Region.

IAM role

Amazon SageMaker requires permissions to call other services. Create a role that has the [AmazonSageMakerFullAccess](#) policy.

AmazonSageMaker-ExecutionRole- XXXXX

## Container definition 1

6. Choose **Create model**.
7. Deploy your multi-model endpoint as you would a single model endpoint. For instructions, see [Deploy the Model to SageMaker Hosting Services \(p. 63\)](#).

## Create a Multi-Model Endpoint (Amazon SDK for Python (Boto))

You create a multi-model endpoint using the Amazon SageMaker `create_model`, `create_endpoint_config`, and `create_endpoint` APIs just as you would create a single model endpoint, but with two changes. When defining the model container, you need to pass a new `Mode` parameter value, `MultiModel`. You also need to pass the `ModelDataUrl` field that specifies the prefix in Amazon S3 where the model artifacts are located, instead of the path to a single model artifact, as you would when deploying a single model.

For a sample notebook that uses SageMaker to deploy multiple XGBoost models to an endpoint, see [Multi-Model Endpoint XGBoost Sample Notebook](#).

The following procedure outlines the key steps used in that sample to create a multi-model endpoint.

### To deploy the model (Amazon SDK for Python (Boto 3))

1. Get a container with an image that supports deploying multi-model endpoints. For a list of built-in algorithms and framework containers that support multi-model endpoints, see [Supported Algorithms and Frameworks \(p. 1896\)](#). For this example, we use the [K-Nearest Neighbors \(k-NN\) Algorithm \(p. 1428\)](#) built-in algorithm. We call the [SageMaker Python SDK](#) utility function `image_uris.retrieve()` to get the address for the K-Nearest Neighbors built-in algorithm image.

```
import sagemaker
region = sagemaker_session.boto_region_name
image = sagemaker.image_uris.retrieve("knn", region=region)
container = { 'Image': image,
              'ModelDataUrl': 's3://my-bucket/path/to/artifacts/',
              'Mode': 'MultiModel'
            }
```

2. Get a Boto 3 SageMaker client and create the model that uses this container.

```
import boto3
sm_client = boto3.client('sagemaker')
response = sm_client.create_model(
    ModelName = 'my-multi-model-name',
    ExecutionRoleArn = role,
    Containers = [container])
```

3. (Optional) If you are using a serial inference pipeline, get the additional container(s) to include in the pipeline, and include it in the `Containers` argument of `CreateModel`:

```
preprocessor_container = { 'Image': '123456789012.dkr.ecr.us-
east-1.amazonaws.com/mypreprocessorimage:mytag'
}

multi_model_container = { 'Image': '763104351884.dkr.ecr.us-
east-1.amazonaws.com/myimage:mytag',
                         'ModelDataUrl': 's3://my-bucket/path/to/artifacts/',
                         'Mode': 'MultiModel'
}

response = sm_client.create_model(
    ModelName = 'my-multi-model-name',
```

```
ExecutionRoleArn = role,
Containers      = [preprocessor_container, multi_model_container])
```

4. (Optional) If your use case does not benefit from model caching, set the value of the ModelCacheSetting field of the MultiModelConfig parameter to Disabled, and include it in the Container argument of the call to `create_model`. The value of the ModelCacheSetting field is Enabled by default.

```
container = {
    'Image': image,
    'ModelDataURL': 's3://my-bucket/path/to/artifacts/',
    'Mode': 'MultiModel',
    'MultiModelConfig': {
        // Default value is 'Enabled'
        'ModelCacheSetting': 'Disabled'
    }
}

response = sm_client.create_model(
    ModelName      = 'my-multi-model-name',
    ExecutionRoleArn = role,
    Containers     = [container])
```

5. Configure the multi-model endpoint for the model. We recommend configuring your endpoints with at least two instances. This allows SageMaker to provide a highly available set of predictions across multiple Availability Zones for the models.

```
response = sm_client.create_endpoint_config(
    EndpointConfigName = 'my-epc',
    ProductionVariants=[{
        'InstanceType':           'ml.m4.xlarge',
        'InitialInstanceCount':  2,
        'InitialVariantWeight':  1,
        'ModelName':              'my-multi-model-name',
        'VariantName':            'AllTraffic'}])
```

#### Note

You can use only one multi-model-enabled endpoint in a serial inference pipeline.

6. Create the multi-model endpoint using the `EndpointName` and `EndpointConfigName` parameters.

```
response = sm_client.create_endpoint(
    EndpointName      = 'my-endpoint',
    EndpointConfigName = 'my-epc')
```

## Invoke a Multi-Model Endpoint

To invoke a multi-model endpoint, use the [invoke\\_endpoint](#) from the SageMaker Runtime just as you would invoke a single model endpoint, with one change. Pass a new `TargetModel` parameter that specifies which of the models at the endpoint to target. The SageMaker Runtime `InvokeEndpoint` request supports `X-Amzn-SageMaker-Target-Model` as a new header that takes the relative path of the model specified for invocation. The SageMaker system constructs the absolute path of the model by combining the prefix that is provided as part of the `CreateModel` API call with the relative path of the model.

The following example prediction request uses the [Amazon SDK for Python \(Boto 3\)](#) in the sample notebook.

```
response = runtime_sm_client.invoke_endpoint(
```

```
EndpointName = 'my-endpoint',
ContentType = 'text/csv',
TargetModel = 'Houston_TX.tar.gz',
Body = body)
```

The multi-model endpoint dynamically loads target models as needed. You can observe this when running the [MME Sample Notebook](#) as it iterates through random invocations against multiple target models hosted behind a single endpoint. The first request against a given model takes longer because the model has to be downloaded from Amazon Simple Storage Service (Amazon S3) and loaded into memory. (This is called a cold start.) Subsequent calls finish faster because there's no additional overhead after the model has loaded.

## Retry Requests on ModelNotReadyException Errors

The first time you call `invoke_endpoint` for a model, the model is downloaded from Amazon Simple Storage Service and loaded into the inference container. This makes the first call take longer to return. Subsequent calls to the same model finish faster, because the model is already loaded.

SageMaker returns a response for a call to `invoke_endpoint` within 60 seconds. Some models are too large to download within 60 seconds. If the model does not finish loading before the 60 second timeout limit, the request to `invoke_endpoint` returns with the error code `ModelNotReadyException`, and the model continues to download and load into the inference container for up to 360 seconds. If you get a `ModelNotReadyException` error code for an `invoke_endpoint` request, retry the request. By default, the Amazon SDKs for Python (Boto 3) (using [Legacy retry mode](#)) and Java retry `invoke_endpoint` requests that result in `ModelNotReadyException` errors. You can configure the retry strategy to continue retrying the request for up to 360 seconds. If you expect your model to take longer than 60 seconds to download and load into the container, set the SDK socket timeout to 70 seconds. For more information about configuring the retry strategy for Boto 3, see [Configuring a retry mode](#). The following code shows an example that configures the retry strategy to retry calls to `invoke_endpoint` for up to 180 seconds.

```
import boto3
from botocore.config import Config

# This example retry strategy sets the retry attempts to 2.
# With this setting, the request can attempt to download and/or load the model
# for upto 180 seconds: 1 orginal request (60 seconds) + 2 retries (120 seconds)
config = Config(
    read_timeout=70,
    retries={
        'max_attempts': 2 # This value can be adjusted to 5 to go up to the 360s max
    timeout
    }
)
runtime_sm_client = boto3.client('sagemaker-runtime', config=config)
```

## Add or Remove Models

You can deploy additional models to a multi-model endpoint and invoke them through that endpoint immediately. When adding a new model, you don't need to update or bring down the endpoint, so you avoid the cost of creating and running a separate endpoint for each new model.

SageMaker unloads unused models from the container when the instance is reaching memory capacity and more models need to be downloaded into the container. SageMaker also deletes unused model artifacts from the instance storage volume when the volume is reaching capacity and new models need to be downloaded. The first invocation to a newly added model takes longer because the endpoint takes time to download the model from S3 to the container's memory in instance hosting the endpoint

With the endpoint already running, copy a new set of model artifacts to the Amazon S3 location there you store your models.

```
# Add an AdditionalModel to the endpoint and exercise it
aws s3 cp AdditionalModel.tar.gz s3://my-bucket/path/to/artifacts/
```

**Important**

To update a model, proceed as you would when adding a new model. Use a new and unique name. Don't overwrite model artifacts in Amazon S3 because the old version of the model might still be loaded in the containers or on the storage volume of the instances on the endpoint. Invocations to the new model could then invoke the old version of the model.

Client applications can request predictions from the additional target model as soon as it is stored in S3.

```
response = runtime_sm_client.invoke_endpoint(
    EndpointName='endpoint_name',
    ContentType='text/csv',
    TargetModel='AdditionalModel.tar.gz',
    Body=body)
```

To delete a model from a multi-model endpoint, stop invoking the model from the clients and remove it from the S3 location where model artifacts are stored.

## Build Your Own Container with Multi Model Server

Custom Elastic Container Registry (ECR) images deployed in Amazon SageMaker are expected to adhere to the basic contract described in [Use Your Own Inference Code with Hosting Services \(p. 2167\)](#) that govern how SageMaker interacts with a Docker container that runs your own inference code. For a container to be capable of loading and serving multiple models concurrently, there are additional APIs and behaviors that must be followed. This additional contract includes new APIs to load, list, get, and unload models, and a different API to invoke models. There are also different behaviors for error scenarios that the APIs need to abide by. To indicate that the container complies with the additional requirements, you can add the following command to your Docker file:

```
LABEL com.amazonaws.sagemaker.capabilities.multi-models=true
```

SageMaker also injects an environment variable into the container

```
SAGEMAKER_MULTI_MODEL=true
```

If you are creating a multi-model endpoint for a serial inference pipeline, your Docker file must have the required labels for both multi-models and serial inference pipelines. For more information about serial information pipelines, see [Run Real-time Predictions with an Inference Pipeline \(p. 1921\)](#).

To help you implement these requirements for a custom container, two libraries are available:

- [Multi Model Server](#) is an open source framework for serving machine learning models that can be installed in containers to provide the front end that fulfills the requirements for the new multi-model endpoint container APIs. It provides the HTTP front end and model management capabilities required by multi-model endpoints to host multiple models within a single container, load models into and unload models out of the container dynamically, and performs inference on a specified loaded model. It also provides a pluggable backend that supports a pluggable custom backend handler where you can implement your own algorithm.
- [SageMaker Inference Toolkit](#) is a library that bootstraps Multi Model Server with a configuration and settings that make it compatible with SageMaker multi-model endpoints. It also allows you to tweak important performance parameters, such as the number of workers per model, depending on the needs of your scenario.

## Use the SageMaker Inference Toolkit

Currently, the only pre-built containers that support multi-model endpoints are the MXNet inference container and the PyTorch inference container. If you want to use any other framework or algorithm, you need to build a container. The easiest way to do this is to use the [SageMaker Inference Toolkit](#) to extend an existing pre-built container. The SageMaker inference toolkit is an implementation for the multi-model server (MMS) that creates endpoints that can be deployed in SageMaker. For a sample notebook that shows how to set up and deploy a custom container that supports multi-model endpoints in SageMaker, see the [Multi-Model Endpoint BYOC Sample Notebook](#).

### Note

The SageMaker inference toolkit supports only Python model handlers. If you want to implement your handler in any other language, you must build your own container that implements the additional multi-model endpoint APIs. For information, see [Contract for Custom Containers to Serve Multiple Models \(p. 1906\)](#).

### To extend a container by using the SageMaker inference toolkit

1. Create a model handler. MMS expects a model handler, which is a Python file that implements functions to pre-process, get predictions from the model, and process the output in a model handler. For an example of a model handler, see [model\\_handler.py](#) from the sample notebook.
2. Import the inference toolkit and use its `model_server.start_model_server` function to start MMS. The following example is from the `dockerd-entrypoint.py` file from the sample notebook. Notice that the call to `model_server.start_model_server` passes the model handler described in the previous step:

```
import subprocess
import sys
import shlex
import os
from retrying import retry
from subprocess import CalledProcessError
from sagemaker_inference import model_server

def _retry_if_error(exception):
    return isinstance(exception, CalledProcessError or OSError)

@retry(stop_max_delay=1000 * 50,
       retry_on_exception=_retry_if_error)
def _start_mms():
    # by default the number of workers per model is 1, but we can configure it through
    # the
    # environment variable below if desired.
    # os.environ['SAGEMAKER_MODEL_SERVER_WORKERS'] = '2'
    model_server.start_model_server(handler_service='/home/model-server/
model_handler.py:handle')

def main():
    if sys.argv[1] == 'serve':
        _start_mms()
    else:
        subprocess.check_call(shlex.split(' '.join(sys.argv[1:])))

    # prevent docker exit
    subprocess.call(['tail', '-f', '/dev/null'])

main()
```

3. In your Dockerfile, copy the model handler from the first step and specify the Python file from the previous step as the entrypoint in your Dockerfile. The following lines are from the [Dockerfile](#) used in the sample notebook:

```
# Copy the default custom service file to handle incoming data and inference requests
COPY model_handler.py /home/model-server/model_handler.py

# Define an entrypoint script for the docker image
ENTRYPOINT [ "python", "/usr/local/bin/dockerd-entrypoint.py" ]
```

4. Build and register your container. The following shell script from the sample notebook builds the container and uploads it to an Amazon Elastic Container Registry repository in your Amazon account:

```
%%sh

# The name of our algorithm
algorithm_name=demo-sagemaker-multimodel

cd container

account=$(aws sts get-caller-identity --query Account --output text)

# Get the region defined in the current configuration (default to us-west-2 if none
# defined)
region=$(aws configure get region)
region=${region:-us-west-2}

fullname="${account}.dkr.ecr.${region}.amazonaws.com/${algorithm_name}:latest"

# If the repository doesn't exist in ECR, create it.
aws ecr describe-repositories --repository-names "${algorithm_name}" > /dev/null 2>&1

if [ $? -ne 0 ]
then
    aws ecr create-repository --repository-name "${algorithm_name}" > /dev/null
fi

# Get the login command from ECR and execute it directly
$(aws ecr get-login --region ${region} --no-include-email)

# Build the docker image locally with the image name and then push it to ECR
# with the full name.

docker build -q -t ${algorithm_name} .
docker tag ${algorithm_name} ${fullname}

docker push ${fullname}
```

You can now use this container to deploy multi-model endpoints in SageMaker.

#### Topics

- [Contract for Custom Containers to Serve Multiple Models \(p. 1906\)](#)

## Contract for Custom Containers to Serve Multiple Models

To handle multiple models, your container must support a set of APIs that enable Amazon SageMaker to communicate with the container for loading, listing, getting, and unloading models as required. The `model_name` is used in the new set of APIs as the key input parameter. The customer container is expected to keep track of the loaded models using `model_name` as the mapping key. Also, the `model_name` is an opaque identifier and is not necessarily the value of the `TargetModel` parameter passed into the `InvokeEndpoint` API. The original `TargetModel` value in the `InvokeEndpoint`

request is passed to container in the APIs as a `X-Amzn-SageMaker-Target-Model` header that can be used for logging purposes.

### Topics

- [Load Model API \(p. 1907\)](#)
- [List Model API \(p. 1907\)](#)
- [Get Model API \(p. 1908\)](#)
- [Unload Model API \(p. 1908\)](#)
- [Invoke Model API \(p. 1908\)](#)

## Load Model API

Instructs the container to load a particular model present in the `url` field of the body into the memory of the customer container and to keep track of it with the assigned `model_name`. After a model is loaded, the container should be ready to serve inference requests using this `model_name`.

```
POST /models HTTP/1.1
Content-Type: application/json
Accept: application/json

{
    "model_name" : "{model_name}",
    "url" : "/opt/ml/models/{model_name}/model",
}
```

### Note

If `model_name` is already loaded, this API should return 409. Any time a model cannot be loaded due to lack of memory or to any other resource, this API should return a 507 HTTP status code to SageMaker, which then initiates unloading unused models to reclaim.

## List Model API

Returns the list of models loaded into the memory of the customer container.

```
GET /models HTTP/1.1
Accept: application/json

Response =
{
    "models": [
        {
            "modelName" : "{model_name}",
            "modelUrl" : "/opt/ml/models/{model_name}/model",
        },
        {
            "modelName" : "{model_name}",
            "modelUrl" : "/opt/ml/models/{model_name}/model",
        },
        ...
    ]
}
```

This API also supports pagination.

```
GET /models HTTP/1.1
Accept: application/json

Response =
```

```
{
  "models": [
    {
      "modelName" : "{model_name}",
      "modelUrl" : "/opt/ml/models/{model_name}/model",
    },
    {
      "modelName" : "{model_name}",
      "modelUrl" : "/opt/ml/models/{model_name}/model",
    },
    ...
  ]
}
```

SageMaker can initially call the List Models API without providing a value for `next_page_token`. If a `nextPageToken` field is returned as part of the response, it will be provided as the value for `next_page_token` in a subsequent List Models call. If a `nextPageToken` is not returned, it means that there are no more models to return.

## Get Model API

This is a simple read API on the `model_name` entity.

```
GET /models/{model_name} HTTP/1.1
Accept: application/json

{
  "modelName" : "{model_name}",
  "modelUrl" : "/opt/ml/models/{model_name}/model",
}
```

### Note

If `model_name` is not loaded, this API should return 404.

## Unload Model API

Instructs the SageMaker platform to instruct the customer container to unload a model from memory. This initiates the eviction of a candidate model as determined by the platform when starting the process of loading a new model. The resources provisioned to `model_name` should be reclaimed by the container when this API returns a response.

```
DELETE /models/{model_name}
```

### Note

If `model_name` is not loaded, this API should return 404.

## Invoke Model API

Makes a prediction request from the particular `model_name` supplied. The SageMaker Runtime `InvokeEndpoint` request supports `X-Amzn-SageMaker-Target-Model` as a new header that takes the relative path of the model specified for invocation. The SageMaker system constructs the absolute path of the model by combining the prefix that is provided as part of the `CreateModel` API call with the relative path of the model.

```
POST /models/{model_name}/invoke HTTP/1.1
Content-Type: ContentType
Accept: Accept
X-Amzn-SageMaker-Custom-Attributes: CustomAttributes
X-Amzn-SageMaker-Target-Model: [relativePath]/[artifactName].tar.gz
```

**Note**

If `model_name` is not loaded, this API should return 404.

## Multi-Model Endpoint Security

Models and data in a multi-model endpoint are co-located on instance storage volume and in container memory. All instances for Amazon SageMaker endpoints run on a single tenant container that you own. Only your models can run on your multi-model endpoint. It's your responsibility to manage the mapping of requests to models and to provide access for users to the correct target models. SageMaker uses [IAM roles](#) to provide IAM identity-based policies that you use to specify allowed or denied actions and resources and the conditions under which actions are allowed or denied.

By default, an IAM principal with `InvokeEndpoint` permissions on a multi-model endpoint can invoke any model at the address of the S3 prefix defined in the `CreateModel` operation, provided that the IAM Execution Role defined in operation has permissions to download the model. If you need to restrict `InvokeEndpoint` access to a limited set of models in S3, you can do one of the following:

- Restrict `InvokeEndpoint` calls to specific models hosted at the endpoint by using the `sagemaker:TargetModel` IAM condition key. For example, the following policy allows `InvokeEndpoint` requests only when the value of the `TargetModel` field matches one of the specified regular expressions:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "sagemaker:InvokeEndpoint"  
            ],  
            "Effect": "Allow",  
            "Resource":  
                "arn:aws:sagemaker:region:account-id:endpoint/endpoint_name",  
            "Condition": {  
                // TargetModel provided must be from this set of values  
                "StringLike": {  
                    "sagemaker:TargetModel": ["company_a/*", "common/*"]  
                }  
            }  
        },  
    ]  
}
```

For information about SageMaker condition keys, see [Condition Keys for Amazon SageMaker](#) in the [Amazon Identity and Access Management User Guide](#).

- Create multi-model endpoints with more restrictive S3 prefixes.

For more information about how SageMaker uses roles to manage access to endpoints and perform operations on your behalf, see [SageMaker Roles \(p. 2447\)](#). Your customers might also have certain data isolation requirements dictated by their own compliance requirements that can be satisfied using IAM identities.

## CloudWatch Metrics for Multi-Model Endpoint Deployments

Amazon SageMaker provides metrics for endpoints so you can monitor the cache hit rate, the number of models loaded, and the model wait times for loading, downloading, and uploading at a multi-

model endpoint. For information, see **Multi-Model Endpoint Model Loading Metrics** and **Multi-Model Endpoint Model Instance Metrics** in [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 2523\)](#). Per-model metrics aren't supported.

## Deploy multi-container endpoints

SageMaker multi-container endpoints enable customers to deploy multiple containers to deploy different models on a SageMaker endpoint. The containers can be run in a sequence as an inference pipeline, or each container can be accessed individually by using direct invocation to improve endpoint utilization and optimize costs.

For information about invoking the containers in a multi-container endpoint in sequence, see [Deploy an Inference Pipeline \(p. 1917\)](#).

For information about invoking a specific container in a multi-container endpoint, see [Use a multi-container endpoint with direct invocation \(p. 1911\)](#)

### Topics

- [Create a multi-container endpoint \(Boto 3\) \(p. 1910\)](#)
- [Update a multi-container endpoint \(p. 1911\)](#)
- [Delete a multi-container endpoint \(p. 1911\)](#)
- [Use a multi-container endpoint with direct invocation \(p. 1911\)](#)
- [Deploy an Inference Pipeline \(p. 1917\)](#)

## Create a multi-container endpoint (Boto 3)

Create a Multi-container endpoint by calling `create_model`, `create_endpoint_config`, and `create_endpoint` APIs as you would to create any other endpoints. You can run these containers sequentially as an inference pipeline, or run each individual container by using direct invocation. Multi-container endpoints have the following requirements when you call `create_model`:

- Use the `Containers` parameter instead of `PrimaryContainer`, and include more than one container in the `Containers` parameter.
- The `ContainerHostname` parameter is required for each container in a multi-container endpoint with direct invocation.
- Set the `Mode` parameter of the `InferenceExecutionConfig` field to `Direct` for direct invocation of each container, or `Serial` to use containers as an inference pipeline. The default mode is `Serial`.

### Note

Currently there is a limit of up to 5 containers supported on a multi-container endpoint.

The following example creates a multi-container model for direct invocation.

1. Create container elements and `InferenceExecutionConfig` with direct invocation.

```
container1 = { 'Image':           '123456789012.dkr.ecr.us-east-1.amazonaws.com/'  
              'myimage1:mytag',  
              'ContainerHostname': 'firstContainer'}  
  
container2 = { 'Image':           '123456789012.dkr.ecr.us-east-1.amazonaws.com/'  
              'myimage2:mytag',  
              'ContainerHostname': 'secondContainer'  
            }  
inferenceExecutionConfig = {'Mode': 'Direct'}
```

}

2. Create the model with the container elements and set the `InferenceExecutionConfig` field.

```
import boto3
sm_client = boto3.Session().client('sagemaker')

response = sm_client.create_model(
    ModelName      = 'my-direct-mode-model-name',
    InferenceExecutionConfig = inferenceExecutionConfig,
    ExecutionRoleArn = role,
    Containers     = [container1, container2])
```

To create an endpoint, you would then call `create_endpoint_config` and `create_endpoint` as you would to create any other endpoint.

## Update a multi-container endpoint

To update a multi-container endpoint, complete the following steps.

1. Call `create_model` to create a new model with a new value for the `Mode` parameter in the `InferenceExecutionConfig` field.
2. Call `create_endpoint_config` to create a new endpoint config with a different name by using the new model you created in the previous step.
3. Call `update_endpoint` to update the endpoint with the new endpoint config you created in the previous step.

## Delete a multi-container endpoint

To delete an endpoint, call `delete_endpoint`, and provide the name of the endpoint you want to delete as the `EndpointName` parameter.

## Use a multi-container endpoint with direct invocation

SageMaker multi-container endpoints enable customers to deploy multiple containers to deploy different models on a SageMaker endpoint. You can host up to 5 different inference containers on a single endpoint. By using direct invocation, you can send a request to a specific inference container hosted on a multi-container endpoint.

### Topics

- [Invoke a multi-container endpoint with direct invocation \(p. 1911\)](#)
- [Security with multi-container endpoints with direct invocation \(p. 1912\)](#)
- [Metrics for multi-container endpoints with direct invocation \(p. 1913\)](#)
- [Autoscale multi-container endpoints \(p. 1916\)](#)
- [Troubleshoot multi-container endpoints \(p. 1916\)](#)

## Invoke a multi-container endpoint with direct invocation

To invoke a multi-container endpoint with direct invocation, call `invoke_endpoint` as you would invoke any other endpoint, and specify which container you want to invoke by using the `TargetContainerHostname` parameter.

The following example directly invokes the `secondContainer` of a multi-container endpoint to get a prediction.

```
import boto3
runtime_sm_client = boto3.Session().client('sagemaker-runtime')

response = runtime_sm_client.invoke_endpoint(
    EndpointName ='my-endpoint',
    ContentType = 'text/csv',
    TargetContainerHostname='secondContainer',
    Body = body)
```

For each direct invocation request to a multi-container endpoint, only the container with the `TargetContainerHostname` processes the invocation request. You will get validation errors if you do any of the following:

- Specify a `TargetContainerHostname` that does not exist in the endpoint
- Do not specify a value for `TargetContainerHostname` in a request to an endpoint configured for direct invocation
- Specify a value for `TargetContainerHostname` in a request to an endpoint that is not configured for direct invocation.

## Security with multi-container endpoints with direct invocation

For multi-container endpoints with direct invocation, there are multiple containers hosted in a single instance by sharing memory and a storage volume. It's your responsibility to use secure containers, maintain the correct mapping of requests to target containers, and provide users with the correct access to target containers. SageMaker uses IAM roles to provide IAM identity-based policies that you use to specify whether access to a resource is allowed or denied to that role, and under what conditions. For information about IAM roles, see [IAM roles in the Amazon Identity and Access Management User Guide](#). For information about identity-based policies, see [Identity-based policies and resource-based policies](#).

By default, an IAM principal with `InvokeEndpoint` permissions on a multi-container endpoint with direct invocation can invoke any container inside the endpoint with the endpoint name that you specify when you call `invoke_endpoint`. If you need to restrict `invoke_endpoint` access to a limited set of containers inside a multi-container endpoint, use the `sagemaker:TargetContainerHostname` IAM condition key. The following policies show how to limit calls to specific containers within an endpoint.

The following policy allows `invoke_endpoint` requests only when the value of the `TargetContainerHostname` field matches one of the specified regular expressions.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sagemaker:InvokeEndpoint"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:sagemaker:region:account-id:endpoint/endpoint_name",
            "Condition": {
                "StringLike": {
                    "sagemaker:TargetContainerHostname": ["customIps*", "common*"]
                }
            }
        ]
    ]
}
```

The following policy denies `invoke_endpoint` requests when the value of the `TargetContainerHostname` field matches one of the specified regular expressions in the `Deny` statement.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "sagemaker:InvokeEndpoint"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:sagemaker:region:account-id:endpoint/endpoint_name",
            "Condition": {
                "StringLike": {
                    "sagemaker:TargetContainerHostname": ["*"]
                }
            }
        },
        {
            "Action": [
                "sagemaker:InvokeEndpoint"
            ],
            "Effect": "Deny",
            "Resource": "arn:aws:sagemaker:region:account-id:endpoint/endpoint_name",
            "Condition": {
                "StringLike": {
                    "sagemaker:TargetContainerHostname": ["special*"]
                }
            }
        }
    ]
}
```

For information about SageMaker condition keys, see [Condition Keys for SageMaker](#) in the *Amazon Identity and Access Management User Guide*.

## Metrics for multi-container endpoints with direct invocation

In addition to the endpoint metrics that are listed in [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 2523\)](#), SageMaker also provides per-container metrics.

Per-container metrics for multi-container endpoints with direct invocation are located in CloudWatch and categorized into two namespaces: `AWS/SageMaker` and `aws/sagemaker/Endpoints`. The `AWS/SageMaker` namespace includes invocation-related metrics, and the `aws/sagemaker/Endpoints` namespace includes memory and CPU utilization metrics.

The following table lists the per-container metrics for multi-container endpoints with direct invocation. All the metrics use the `[EndpointName, VariantName, ContainerName]` dimension, which filters metrics at a specific endpoint, for a specific variant and corresponding to a specific container. These metrics share the same metric names as in those for inference pipelines, but at a per-container level `[EndpointName, VariantName, ContainerName]`.

Metric Name	Description	Dimension	NameSpace
<b>Invocations</b>	The number of <code>InvokeEndpoint</code> requests sent to a container inside an	<code>EndpointName,</code> <code>VariantName,</code> <code>ContainerName</code>	<code>AWS/SageMaker</code>

	endpoint. To get the total number of requests sent to that container, use the Sum statistic. Units: None Valid statistics: Sum, Sample Count		
Invocation4XX Errors	The number of InvokeEndpoint requests that the model returned a 4xx HTTP response code for on a specific container. For each 4xx response, SageMaker sends a 1. Units: None Valid statistics: Average, Sum	EndpointName, VariantName, ContainerName	AWS/SageMaker
Invocation5XX Errors	The number of InvokeEndpoint requests that the model returned a 5xx HTTP response code for on a specific container. For each 5xx response, SageMaker sends a 1. Units: None Valid statistics: Average, Sum	EndpointName, VariantName, ContainerName	AWS/SageMaker
ContainerLatency	The time it took for the target container to respond as viewed from SageMaker. ContainerLatency includes the time it took to send the request, to fetch the response from the model's container, and to complete inference in the container. Units: Microseconds Valid statistics: Average, Sum, Min, Max, Sample Count	EndpointName, VariantName, ContainerName	AWS/SageMaker

OverheadLatency	The time added to the time taken to respond to a client request by SageMaker for overhead. OverheadLatency is measured from the time that SageMaker receives the request until it returns a response to the client, minus theModelLatency. Overhead latency can vary depending on request and response payload sizes, request frequency, and authentication or authorization of the request, among other factors. Units: Microseconds Valid statistics: Average, Sum, Min, Max, `Sample Count`	EndpointName, VariantName, ContainerName	AWS/SageMaker
CPUUtilization	The percentage of CPU units that are used by each container running on an instance. The value ranges from 0% to 100%, and is multiplied by the number of CPUs. For example, if there are four CPUs, CPUUtilization can range from 0% to 400%. For endpoints with direct invocation, the number of CPUUtilization metrics equals the number of containers in that endpoint. Units: Percent	EndpointName, VariantName, ContainerName	aws/sagemaker/Endpoints

<code>MemoryUtilizaton</code>	The percentage of memory that is used by each container running on an instance. This value ranges from 0% to 100%. Similar as <code>CPUUtilization</code> , in endpoints with direct invocation, the number of <code>MemoryUtilization</code> metrics equals the number of containers in that endpoint. Units: Percent	<code>EndpointName</code> , <code>VariantName</code> , <code>ContainerName</code>	<code>aws/sagemaker/Endpoints</code>
-------------------------------	--	---	--------------------------------------

All the metrics in the previous table are specific to multi-container endpoints with direct invocation. Besides these special per-container metrics, there are also metrics at the variant level with dimension [`EndpointName`, `VariantName`] for all the metrics in the table expect `ContainerLatency`.

## Autoscale multi-container endpoints

If you want to configure automatic scaling for a multi-container endpoint using the `InvocationsPerInstance` metric, we recommend that the model in each container exhibits similar CPU utilization and latency on each inference request. This is recommended because if traffic to the multi-container endpoint shifts from a high CPU utilization model to a low CPU utilization model, but the overall call volume remains the same, the endpoint does not scale out and there may not be enough instances to handle all the requests to the high CPU utilization model. For information about automatically scaling endpoints, see [Automatically Scale Amazon SageMaker Models \(p. 1931\)](#).

## Troubleshoot multi-container endpoints

The following sections can help you troubleshoot errors with multi-container endpoints.

### Ping Health Check Errors

With multiple containers, endpoint memory and CPU are under higher pressure during endpoint creation. Specifically, the `MemoryUtilization` and `CPUUtilization` metrics are higher than for single-container endpoints, because utilization pressure is proportional to the number of containers. Because of this, we recommend that you choose instance types with enough memory and CPU to ensure that there is enough memory on the instance to have all the models loaded (the same guidance applies to deploying an inference pipeline). Otherwise, your endpoint creation might fail with an error such as `XXX did not pass the ping health check`.

### Missing accept-bind-to-port=true Docker label

The containers in a multi-container endpoints listen on the port specified in the `SAGEMAKER_BIND_TO_PORT` environment variable instead of port 8080. When a container runs in a multi-container endpoint, SageMaker automatically provides this environment variable to the container. If this environment variable isn't present, containers default to using port 8080. To indicate that your container complies with this requirement, use the following command to add a label to your Dockerfile:

```
LABEL com.amazonaws.sagemaker.capabilities.accept-bind-to-port=true
```

Otherwise, You will see an error message such as `Your Ecr Image XXX does not contain required com.amazonaws.sagemaker.capabilities.accept-bind-to-port=true Docker label(s).`

If your container needs to listen on a second port, choose a port in the range specified by the `SAGEMAKER_SAFE_PORT_RANGE` environment variable. Specify the value as an inclusive range in the format `XXXX-YYYY`, where XXXX and YYYY are multi-digit integers. SageMaker provides this value automatically when you run the container in a multi-container endpoint.

## Deploy an Inference Pipeline

An *inference pipeline* is a Amazon SageMaker model that is composed of a linear sequence of two to five containers that process requests for inferences on data. You use an inference pipeline to define and deploy any combination of pretrained SageMaker built-in algorithms and your own custom algorithms packaged in Docker containers. You can use an inference pipeline to combine preprocessing, predictions, and post-processing data science tasks. Inference pipelines are fully managed.

You can add SageMaker Spark ML Serving and scikit-learn containers that reuse the data transformers developed for training models. The entire assembled inference pipeline can be considered as a SageMaker model that you can use to make either real-time predictions or to process batch transforms directly without any external preprocessing.

Within an inference pipeline model, SageMaker handles invocations as a sequence of HTTP requests. The first container in the pipeline handles the initial request, then the intermediate response is sent as a request to the second container, and so on, for each container in the pipeline. SageMaker returns the final response to the client.

When you deploy the pipeline model, SageMaker installs and runs all of the containers on each Amazon Elastic Compute Cloud (Amazon EC2) instance in the endpoint or transform job. Feature processing and inferences run with low latency because the containers are co-located on the same EC2 instances. You define the containers for a pipeline model using the [CreateModel](#) operation or from the console. Instead of setting one `PrimaryContainer`, you use the `Containers` parameter. to set the containers that make up the pipeline You also specify the order in which the containers are executed.

A pipeline model is immutable, but you can update an inference pipeline by deploying a new one using the [UpdateEndpoint](#) operation. This modularity supports greater flexibility during experimentation.

There are no additional costs for using this feature. You pay only for the instances running on an endpoint.

### Topics

- [Sample Notebooks for Inference Pipelines \(p. 1917\)](#)
- [Feature Processing with Spark ML and Scikit-learn \(p. 1918\)](#)
- [Create a Pipeline Model \(p. 1919\)](#)
- [Run Real-time Predictions with an Inference Pipeline \(p. 1921\)](#)
- [Run Batch Transforms with Inference Pipelines \(p. 1923\)](#)
- [Inference Pipeline Logs and Metrics \(p. 1924\)](#)
- [Troubleshoot Inference Pipelines \(p. 1929\)](#)

## Sample Notebooks for Inference Pipelines

For a sample notebook that uploads and processes a dataset, trains a model, and builds a pipeline model, see the [Inference Pipelines with Spark ML and XGBoost on Abalone](#) notebook. This notebook shows how you can build your machine learning pipeline by using Spark feature Transformers and the

SageMaker XGBoost algorithm. After training the model, the sample shows how to deploy the pipeline (feature Transformer and XGBoost) for real-time predictions and also performs a batch transform job using the same pipeline.

For an example that shows how to do both pre-processing and post-processing by using an inference pipeline, see [Deploy Apache Spark pre-processing and post-processing with XGBoost for real-time prediction requests in Amazon SageMaker using Inference Pipelines](#)

For more examples that show how to create and deploy inference pipelines, see the [Inference Pipelines with SparkML and BlazingText on DBpedia](#) and [Training using SparkML on EMR and hosting on SageMaker](#) sample notebooks. For instructions on creating and accessing Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#).

To see a list of all the SageMaker samples, after creating and opening a notebook instance, choose the **SageMaker Examples** tab. There are three inference pipeline notebooks. The first two inference pipeline notebooks just described are located in the advanced\_functionality folder and the third notebook is in the sagemaker-python-sdk folder. To open a notebook, choose its **Use** tab, then choose **Create copy**.

## Feature Processing with Spark ML and Scikit-learn

Before training a model with either Amazon SageMaker built-in algorithms or custom algorithms, you can use Spark and scikit-learn preprocessors to transform your data and engineer features.

### Feature Processing with Spark ML

You can run Spark ML jobs with [Amazon Glue](#), a serverless ETL (extract, transform, load) service, from your SageMaker notebook. You can also connect to existing EMR clusters to run Spark ML jobs with [Amazon EMR](#). To do this, you need an Amazon Identity and Access Management (IAM) role that grants permission for making calls from your SageMaker notebook to Amazon Glue.

#### Note

To see which Python and Spark versions Amazon Glue supports, refer to [Amazon Glue Release Notes](#).

After engineering features, you package and serialize Spark ML jobs with MLeap into MLeap containers that you can add to an inference pipeline. You don't need to use externally managed Spark clusters. With this approach, you can seamlessly scale from a sample of rows to terabytes of data. The same transformers work for both training and inference, so you don't need to duplicate preprocessing and feature engineering logic or develop a one-time solution to make the models persist. With inference pipelines, you don't need to maintain outside infrastructure, and you can make predictions directly from data inputs.

When you run a Spark ML job on Amazon Glue, a Spark ML pipeline is serialized into [MLeap](#) format. Then, you can use the job with the [SparkML Model Serving Container](#) in a SageMaker Inference Pipeline. [MLeap](#) is a serialization format and execution engine for machine learning pipelines. It supports Spark, Scikit-learn, and TensorFlow for training pipelines and exporting them to a serialized pipeline called an MLeap Bundle. You can deserialize Bundles back into Spark for batch-mode scoring or into the MLeap runtime to power real-time API services.

### Feature Processing with Sci-kit Learn

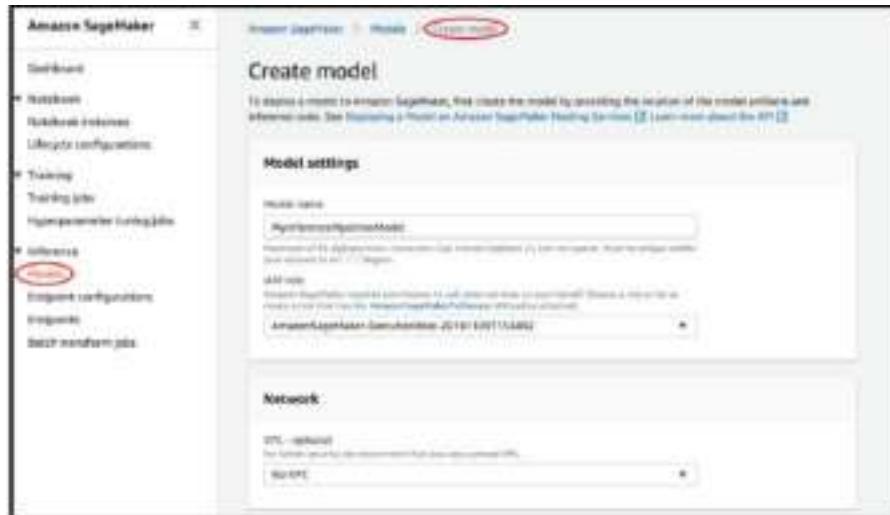
You can run and package scikit-learn jobs into containers directly in Amazon SageMaker. For an example of Python code for building a scikit-learn featurizer model that trains on [Fisher's Iris flower data set](#) and predicts the species of Iris based on morphological measurements, see [IRIS Training and Prediction with Sagemaker Scikit-learn](#).

## Create a Pipeline Model

To create a pipeline model that can be deployed to an endpoint or used for a batch transform job, use the Amazon SageMaker console or the `CreateModel` operation.

### To create an inference pipeline (console)

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Choose **Models**, and then choose **Create models** from the **Inference** group.
3. On the **Create model** page, provide a model name, choose an IAM role, and, if you want to use a private VPC, specify VPC values.



4. To add information about the containers in the inference pipeline, choose **Add container**, then choose **Next**.
5. Complete the fields for each container in the order that you want to execute them, up to the maximum of five. Complete the **Container input options**, **Location of inference code image**, and, optionally, **Location of model artifacts**, **Container host name**, and **Environmental variables** fields.

**Container definition 1**

\* Container input options

Provide model artifacts and inference image.

\* Provide model artifacts and inference image

Location of inference code image

The required code when my inference code image is served in Amazon ECR.

Location of model artifacts - optional

The URL for the S3 location where model artifacts are stored.

The path must point to a single zip component for inference image artifacts.

Container host name - optional

The DNS host name for the container.

Metadata of ET attachments (Resource, Data, Model, Pipeline, Job) is saved within your account from AWS Regions.

\* Environment variables - optional

Key	Value	Remove
key1	value1	<input type="button" value="Remove"/>
key2	value2	<input type="button" value="Remove"/>

Add environment variable

**Container definition 2 - optional**

\* Container input options

Provide model artifacts and inference image.

\* Provide model artifacts and inference image

Location of inference code image

The required code when my inference code image is served in Amazon ECR.

Location of model artifacts - optional

The URL for the S3 location where model artifacts are stored.

The path must point to a single zip component for inference image artifacts.

Container host name - optional

The DNS host name for the container.

Metadata of ET attachments (Resource, Data, Model, Pipeline, Job) is saved within your account from AWS Regions.

\* Environment variables - optional

Key	Value	Remove
		<input type="button" value="Remove"/>

Add environment variable

**Container definition 3 - optional**

\* Container input options

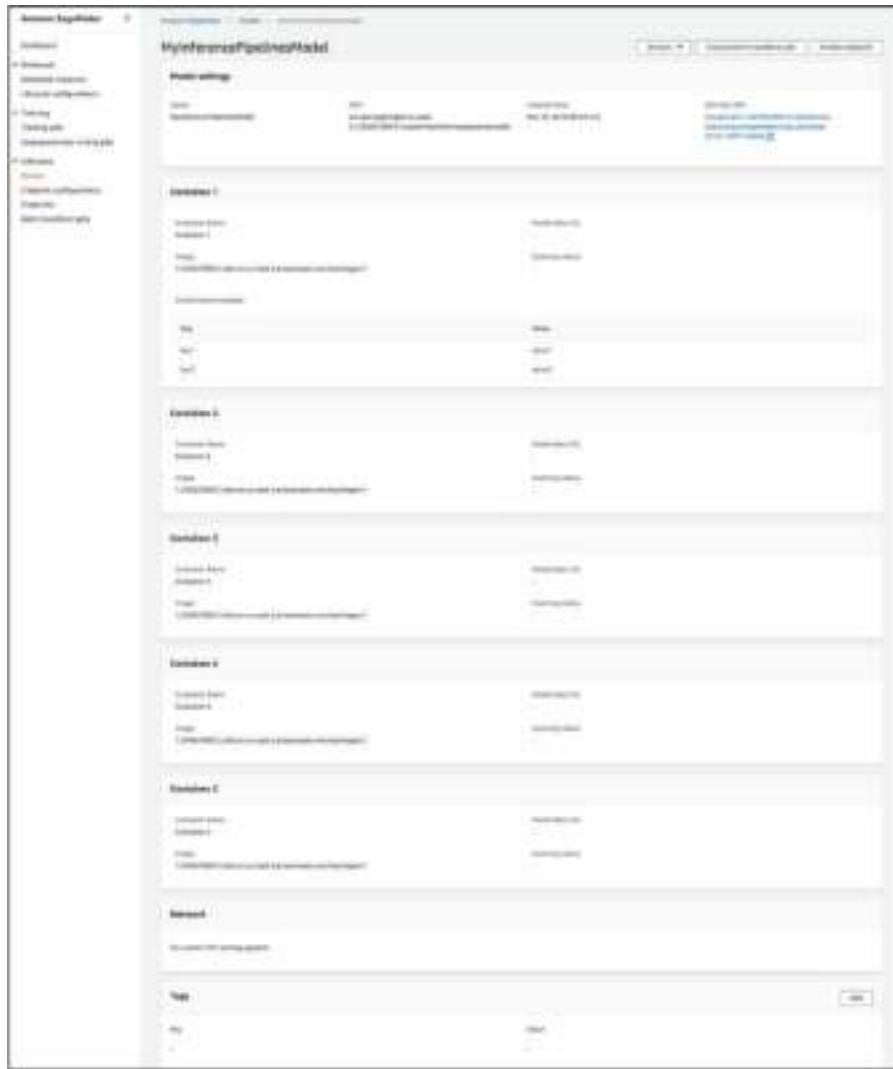
Provide model artifacts and inference image.

\* Provide model artifacts and inference image

Location of inference code image

The required code when my inference code image is served in Amazon ECR.

The **MyInferencePipelineModel** page summarizes the settings for the containers that provide input for the model. If you provided the environment variables in a corresponding container definition, SageMaker shows them in the **Environment variables** field.



## Run Real-time Predictions with an Inference Pipeline

You can use trained models in an inference pipeline to make real-time predictions directly without performing external preprocessing. When you configure the pipeline, you can choose to use the built-in feature transformers already available in Amazon SageMaker. Or, you can implement your own transformation logic using just a few lines of scikit-learn or Spark code.

[MLeap](#), a serialization format and execution engine for machine learning pipelines, supports Spark, scikit-learn, and TensorFlow for training pipelines and exporting them to a serialized pipeline called an MLeap Bundle. You can deserialize Bundles back into Spark for batch-mode scoring or into the MLeap runtime to power real-time API services.

The containers in a pipeline listen on the port specified in the `SAGEMAKER_BIND_TO_PORT` environment variable (instead of 8080). When running in an inference pipeline, SageMaker automatically provides this environment variable to containers. If this environment variable isn't present, containers default to using

port 8080. To indicate that your container complies with this requirement, use the following command to add a label to your Dockerfile:

```
LABEL com.amazonaws.sagemaker.capabilities.accept-bind-to-port=true
```

If your container needs to listen on a second port, choose a port in the range specified by the `SAGEMAKER_SAFE_PORT_RANGE` environment variable. Specify the value as an inclusive range in the format "`XXXX-YYYY`", where `XXXX` and `YYYY` are multi-digit integers. SageMaker provides this value automatically when you run the container in a multicontainer pipeline.

**Note**

To use custom Docker images in a pipeline that includes [SageMaker built-in algorithms](#), you need an [Amazon Elastic Container Registry \(Amazon ECR\) policy](#). Your Amazon ECR repository must grant SageMaker permission to pull the image. For more information, see [Troubleshoot Amazon ECR Permissions for Inference Pipelines \(p. 1929\)](#).

## Create and Deploy an Inference Pipeline Endpoint

The following code creates and deploys a real-time inference pipeline model with SparkML and XGBoost models in series using the SageMaker SDK.

```
from sagemaker.model import Model
from sagemaker.pipeline_model import PipelineModel
from sagemaker.sparkml.model import SparkMLModel

sparkml_data = 's3://{{}/{}}'.format(s3_model_bucket, s3_model_key_prefix, 'model.tar.gz')
sparkml_model = SparkMLModel(model_data=sparkml_data)
xgb_model = Model(model_data=xgb_model.model_data, image=training_image)

model_name = 'serial-inference-' + timestamp_prefix
endpoint_name = 'serial-inference-ep-' + timestamp_prefix
sm_model = PipelineModel(name=model_name, role=role, models=[sparkml_model, xgb_model])
sm_model.deploy(initial_instance_count=1, instance_type='ml.c4.xlarge',
    endpoint_name=endpoint_name)
```

## Request Real-Time Inference from an Inference Pipeline Endpoint

The following example shows how to make real-time predictions by calling an inference endpoint and passing a request payload in JSON format:

```
import sagemaker
from sagemaker.predictor import json_serializer, json_deserializer, Predictor

payload = {
    "input": [
        {
            "name": "Pclass",
            "type": "float",
            "val": "1.0"
        },
        {
            "name": "Embarked",
            "type": "string",
            "val": "Q"
        },
        {
            "name": "Age",
            "type": "double",
            "val": "48.0"
        }
    ]
}
```

```

        "name": "Fare",
        "type": "double",
        "val": "100.67"
    },
    {
        "name": "SibSp",
        "type": "double",
        "val": "1.0"
    },
    {
        "name": "Sex",
        "type": "string",
        "val": "male"
    }
],
"output": [
    {
        "name": "features",
        "type": "double",
        "struct": "vector"
    }
]
}

predictor = Predictor(endpoint=endpoint_name, sagemaker_session=sagemaker.Session(),
    serializer=json_serializer,
    content_type='text/csv', accept='application/json'

print(predictor.predict(payload))

```

The response you get from `predictor.predict(payload)` is the model's inference result.

### Realtime inference pipeline example

You can run this [example notebook using the SKLearn predictor](#) that shows how to deploy an endpoint, run an inference request, then deserialize the response. Find this notebook and more examples in the [Amazon SageMaker example GitHub repository](#).

## Run Batch Transforms with Inference Pipelines

To get inferences on an entire dataset you run a batch transform on a trained model. To run inferences on a full dataset, you can use the same inference pipeline model created and deployed to an endpoint for real-time processing in a batch transform job. To run a batch transform job in a pipeline, you download the input data from Amazon S3 and send it in one or more HTTP requests to the inference pipeline model. For an example that shows how to prepare data for a batch transform, see the "Preparing Data for Batch Transform" section of the [ML Pipeline with SparkML and XGBoost - Training and Inference sample notebook](#). For information about Amazon SageMaker batch transforms, see [Get Inferences for an Entire Dataset with Batch Transform \(p. 13\)](#).

#### Note

To use custom Docker images in a pipeline that includes [Amazon SageMaker built-in algorithms](#), you need an [Amazon Elastic Container Registry \(ECR\) policy](#). Your Amazon ECR repository must grant SageMaker permission to pull the image. For more information, see [Troubleshoot Amazon ECR Permissions for Inference Pipelines \(p. 1929\)](#).

The following example shows how to run a transform job using the [Amazon SageMaker Python SDK](#). In this example, `model1_name` is the inference pipeline that combines SparkML and XGBoost models (created in previous examples). The Amazon S3 location specified by `input_data_path` contains the input data, in CSV format, to be downloaded and sent to the Spark ML model. After the transform job has finished, the Amazon S3 location specified by `output_data_path` contains the output data returned by the XGBoost model in CSV format.

```

import sagemaker
input_data_path = 's3://{}//{}//{}'.format(default_bucket, 'key', 'file_name')

```

```

output_data_path = 's3://{}{}'.format(default_bucket, 'key')
transform_job = sagemaker.transformer.Transformer(
    model_name = model_name,
    instance_count = 1,
    instance_type = 'ml.m4.xlarge',
    strategy = 'SingleRecord',
    assemble_with = 'Line',
    output_path = output_data_path,
    base_transform_job_name='inference-pipelines-batch',
    sagemaker_session=sagemaker.Session(),
    accept = CONTENT_TYPE_CSV)
transform_job.transform(data = input_data_path,
                       content_type = CONTENT_TYPE_CSV,
                       split_type = 'Line')

```

## Inference Pipeline Logs and Metrics

Monitoring is important for maintaining the reliability, availability, and performance of Amazon SageMaker resources. To monitor and troubleshoot inference pipeline performance, use Amazon CloudWatch logs and error messages. For information about the monitoring tools that SageMaker provides, see [Monitor Amazon SageMaker \(p. 2523\)](#).

### Use Metrics to Monitor Multi-container Models

To monitor the multi-container models in Inference Pipelines, use Amazon CloudWatch. CloudWatch collects raw data and processes it into readable, near real-time metrics. SageMaker training jobs and endpoints write CloudWatch metrics and logs in the AWS/SageMaker namespace.

The following tables list the metrics and dimensions for the following:

- Endpoint invocations
- Training jobs, batch transform jobs, and endpoint instances

A *dimension* is a name/value pair that uniquely identifies a metric. You can assign up to 10 dimensions to a metric. For more information on monitoring with CloudWatch, see [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 2523\)](#).

#### Endpoint Invocation Metrics

The AWS/SageMaker namespace includes the following request metrics from calls to [InvokeEndpoint](#).

Metrics are reported at a 1-minute intervals.

Metric	Description
<code>Invocation4XXErrors</code>	The number of <code>InvokeEndpoint</code> requests that the model returned a 4xx HTTP response code for. For each 4xx response, SageMaker sends a 1.  Units: None  Valid statistics: Average, Sum
<code>Invocation5XXErrors</code>	The number of <code>InvokeEndpoint</code> requests that the model returned a 5xx HTTP response code for. For each 5xx response, SageMaker sends a 1.  Units: None  Valid statistics: Average, Sum

Metric	Description
Invocations	<p>The number of <code>InvokeEndpoint</code> requests sent to a model endpoint.</p> <p>To get the total number of requests sent to a model endpoint, use the <code>Sum</code> statistic.</p> <p>Units: None</p> <p>Valid statistics: <code>Sum</code>, <code>Sample Count</code></p>
InvocationsPerInstance	<p>The number of endpoint invocations sent to a model, normalized by <code>InstanceCount</code> in each <code>ProductionVariant</code>. SageMaker sends <math>1/\text{numberOfInstances}</math> as the value for each request, where <code>numberOfInstances</code> is the number of active instances for the <code>ProductionVariant</code> at the endpoint at the time of the request.</p> <p>Units: None</p> <p>Valid statistics: <code>Sum</code></p>
ModelLatency	<p>The time the model or models took to respond. This includes the time it took to send the request, to fetch the response from the model container, and to complete the inference in the container. <code>ModelLatency</code> is the total time taken by all containers in an inference pipeline.</p> <p>Units: Microseconds</p> <p>Valid statistics: <code>Average</code>, <code>Sum</code>, <code>Min</code>, <code>Max</code>, <code>Sample Count</code></p>
OverheadLatency	<p>The time added to the time taken to respond to a client request by SageMaker for overhead. <code>OverheadLatency</code> is measured from the time that SageMaker receives the request until it returns a response to the client, minus the <code>ModelLatency</code>. Overhead latency can vary depending on request and response payload sizes, request frequency, and authentication or authorization of the request, among other factors.</p> <p>Units: Microseconds</p> <p>Valid statistics: <code>Average</code>, <code>Sum</code>, <code>Min</code>, <code>Max</code>, <code>Sample Count</code></p>
ContainerLatency	<p>The time it took for an Inference Pipelines container to respond as viewed from SageMaker. <code>ContainerLatency</code> includes the time it took to send the request, to fetch the response from the model's container, and to complete inference in the container.</p> <p>Units: Microseconds</p> <p>Valid statistics: <code>Average</code>, <code>Sum</code>, <code>Min</code>, <code>Max</code>, <code>Sample Count</code></p>

#### Dimensions for Endpoint Invocation Metrics

Dimension	Description
<code>EndpointName</code> , <code>VariantName</code> , <code>ContainerName</code>	Filters endpoint invocation metrics for a <code>ProductionVariant</code> at the specified endpoint and for the specified variant.

For an inference pipeline endpoint, CloudWatch lists per-container latency metrics in your account as **Endpoint Container Metrics** and **Endpoint Variant Metrics** in the **SageMaker** namespace, as follows. The **ContainerLatency** metric appears only for inferences pipelines.



For each endpoint and each container, latency metrics display names for the container, endpoint, variant, and metric.



#### Training Job, Batch Transform Job, and Endpoint Instance Metrics

The namespaces `/aws/sagemaker/TrainingJobs`, `/aws/sagemaker/TransformJobs`, and `/aws/sagemaker/Endpoints` include the following metrics for training jobs and endpoint instances.

Metrics are reported at a 1-minute intervals.

Metric	Description
CPUUtilization	<p>The percentage of CPU units that are used by the containers running on an instance. The value ranges from 0% to 100%, and is multiplied by the number of CPUs. For example, if there are four CPUs, <b>CPUUtilization</b> can range from 0% to 400%.</p> <p>For training jobs, <b>CPUUtilization</b> is the CPU utilization of the algorithm container running on the instance.</p> <p>For batch transform jobs, <b>CPUUtilization</b> is the CPU utilization of the transform container running on the instance.</p> <p>For multi-container models, <b>CPUUtilization</b> is the sum of CPU utilization by all containers running on the instance.</p> <p>For endpoint variants, <b>CPUUtilization</b> is the sum of CPU utilization by all of the containers running on the instance.</p> <p>Units: Percent</p>
MemoryUtilization	<p>The percentage of memory that is used by the containers running on an instance. This value ranges from 0% to 100%.</p> <p>For training jobs, <b>MemoryUtilization</b> is the memory used by the algorithm container running on the instance.</p> <p>For batch transform jobs, <b>MemoryUtilization</b> is the memory used by the transform container running on the instance.</p>

Metric	Description
	<p>For multi-container models, <code>MemoryUtilization</code> is the sum of memory used by all containers running on the instance.</p> <p>For endpoint variants, <code>MemoryUtilization</code> is the sum of memory used by all of the containers running on the instance.</p> <p>Units: Percent</p>
<code>GPUUtilization</code>	<p>The percentage of GPU units that are used by the containers running on an instance. <code>GPUUtilization</code> ranges from 0% to 100% and is multiplied by the number of GPUs. For example, if there are four GPUs, <code>GPUUtilization</code> can range from 0% to 400%.</p> <p>For training jobs, <code>GPUUtilization</code> is the GPU used by the algorithm container running on the instance.</p> <p>For batch transform jobs, <code>GPUUtilization</code> is the GPU used by the transform container running on the instance.</p> <p>For multi-container models, <code>GPUUtilization</code> is the sum of GPU used by all containers running on the instance.</p> <p>For endpoint variants, <code>GPUUtilization</code> is the sum of GPU used by all of the containers running on the instance.</p> <p>Units: Percent</p>
<code>GPUMemoryUtilization</code>	<p>The percentage of GPU memory used by the containers running on an instance. <code>GPUMemoryUtilization</code> ranges from 0% to 100% and is multiplied by the number of GPUs. For example, if there are four GPUs, <code>GPUMemoryUtilization</code> can range from 0% to 400%.</p> <p>For training jobs, <code>GPUMemoryUtilization</code> is the GPU memory used by the algorithm container running on the instance.</p> <p>For batch transform jobs, <code>GPUMemoryUtilization</code> is the GPU memory used by the transform container running on the instance.</p> <p>For multi-container models, <code>GPUMemoryUtilization</code> is sum of GPU used by all containers running on the instance.</p> <p>For endpoint variants, <code>GPUMemoryUtilization</code> is the sum of the GPU memory used by all of the containers running on the instance.</p> <p>Units: Percent</p>
<code>DiskUtilization</code>	<p>The percentage of disk space used by the containers running on an instance. <code>DiskUtilization</code> ranges from 0% to 100%. This metric is not supported for batch transform jobs.</p> <p>For training jobs, <code>DiskUtilization</code> is the disk space used by the algorithm container running on the instance.</p> <p>For endpoint variants, <code>DiskUtilization</code> is the sum of the disk space used by all of the provided containers running on the instance.</p> <p>Units: Percent</p>

### Dimensions for Training Job, Batch Transform Job, and Endpoint Instance Metrics

Dimension	Description
Host	<p>For training jobs, Host has the format [training-job-name]/algo-[instance-number-in-cluster]. Use this dimension to filter instance metrics for the specified training job and instance. This dimension format is present only in the /aws/sagemaker/TrainingJobs namespace.</p> <p>For batch transform jobs, Host has the format [transform-job-name]/[instance-id]. Use this dimension to filter instance metrics for the specified batch transform job and instance. This dimension format is present only in the /aws/sagemaker/TransformJobs namespace.</p> <p>For endpoints, Host has the format [endpoint-name]/[production-variant-name]/[instance-id]. Use this dimension to filter instance metrics for the specified endpoint, variant, and instance. This dimension format is present only in the /aws/sagemaker/Endpoints namespace.</p>

To help you debug your training jobs, endpoints, and notebook instance lifecycle configurations, SageMaker also sends anything an algorithm container, a model container, or a notebook instance lifecycle configuration sends to `stdout` or `stderr` to Amazon CloudWatch Logs. You can use this information for debugging and to analyze progress.

### Use Logs to Monitor an Inference Pipeline

The following table lists the log groups and log streams SageMaker sends to Amazon CloudWatch

A *log stream* is a sequence of log events that share the same source. Each separate source of logs into CloudWatch makes up a separate log stream. A *log group* is a group of log streams that share the same retention, monitoring, and access control settings.

#### Logs

Log Group Name	Log Stream Name
/aws/sagemaker/TrainingJobs	[training-job-name]/algo-[instance-number-in-cluster]-[epoch_timestamp]
/aws/sagemaker/Endpoints/[EndpointName]	[production-variant-name]/[instance-id] [production-variant-name]/[instance-id] [production-variant-name]/[instance-id]/[container-name provided in the SageMaker model] (For Inference Pipelines) For Inference Pipelines logs, if you don't provide container names, CloudWatch uses **container-1, container-2**, and so on, in the order that containers are provided in the model.
/aws/sagemaker/NotebookInstances	[notebook-instance-name]/[LifecycleConfigHook]
/aws/sagemaker/TransformJobs	[transform-job-name]/[instance-id]-[epoch_timestamp] [transform-job-name]/[instance-id]-[epoch_timestamp]/data-log

Log Group Name	Log Stream Name
	[transform-job-name]/[instance-id]-[epoch_timestamp]/[container-name provided in the SageMaker model] (For Inference Pipelines) For Inference Pipelines logs, if you don't provide container names, CloudWatch uses **container-1, container-2**, and so on, in the order that containers are provided in the model.

**Note**

SageMaker creates the /aws/sagemaker/NotebookInstances log group when you create a notebook instance with a lifecycle configuration. For more information, see [Customize a Notebook Instance Using a Lifecycle Configuration Script \(p. 124\)](#).

For more information about SageMaker logging, see [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 2534\)](#).

## Troubleshoot Inference Pipelines

To troubleshoot inference pipeline issues, use CloudWatch logs and error messages. If you are using custom Docker images in a pipeline that includes Amazon SageMaker built-in algorithms, you might also encounter permissions problems. To grant the required permissions, create an Amazon Elastic Container Registry (Amazon ECR) policy.

**Topics**

- [Troubleshoot Amazon ECR Permissions for Inference Pipelines \(p. 1929\)](#)
- [Use CloudWatch Logs to Troubleshoot SageMaker Inference Pipelines \(p. 1929\)](#)
- [Use Error Messages to Troubleshoot Inference Pipelines \(p. 1930\)](#)

## Troubleshoot Amazon ECR Permissions for Inference Pipelines

When you use custom Docker images in a pipeline that includes [SageMaker built-in algorithms](#), you need an [Amazon ECR policy](#). The policy allows your Amazon ECR repository to grant permission for SageMaker to pull the image. The policy must add the following permissions:

```
{
    "Version": "2008-10-17",
    "Statement": [
        {
            "Sid": "allowSageMakerToPull",
            "Effect": "Allow",
            "Principal": {
                "Service": "sagemaker.amazonaws.com"
            },
            "Action": [
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability"
            ]
        }
    ]
}
```

## Use CloudWatch Logs to Troubleshoot SageMaker Inference Pipelines

SageMaker publishes the container logs for endpoints that deploy an inference pipeline to Amazon CloudWatch at the following path for each container.

```
/aws/sagemaker/Endpoints/{EndpointName}/{Variant}/{InstanceId}/{ContainerHostname}
```

For example, logs for this endpoint are published to the following log groups and streams:

```
EndpointName: MyInferencePipelinesEndpoint
Variant: MyInferencePipelinesVariant
InstanceId: i-0179208609ff7e488
ContainerHostname: MyContainerName1 and MyContainerName2
```

```
logGroup: /aws/sagemaker/Endpoints/MyInferencePipelinesEndpoint
logStream: MyInferencePipelinesVariant/i-0179208609ff7e488/MyContainerName1
logStream: MyInferencePipelinesVariant/i-0179208609ff7e488/MyContainerName2
```

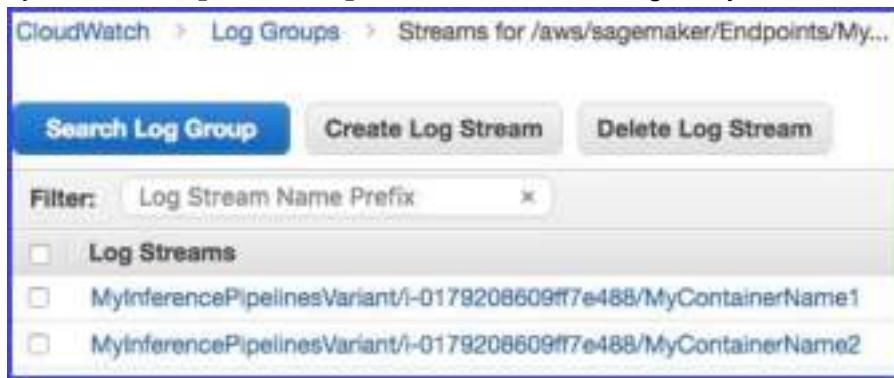
A *log stream* is a sequence of log events that share the same source. Each separate source of logs into CloudWatch makes up a separate log stream. A *log group* is a group of log streams that share the same retention, monitoring, and access control settings.

### To see the log groups and streams

1. Open the CloudWatch console at <https://console.amazonaws.cn/cloudwatch/>.
2. In the navigation page, choose **Logs**.
3. In **Log Groups**, filter on **MyInferencePipelinesEndpoint**:



4. To see the log streams, on the CloudWatch **Log Groups** page, choose **MyInferencePipelinesEndpoint**, and then **Search Log Group**.



For a list of the logs that SageMaker publishes, see [Inference Pipeline Logs and Metrics \(p. 1924\)](#).

### Use Error Messages to Troubleshoot Inference Pipelines

The inference pipeline error messages indicate which containers failed.

If an error occurs while SageMaker is invoking an endpoint, the service returns a `ModelError` (error code 424), which indicates which container failed. If the request payload (the response from the previous container) exceeds the limit of 5 MB, SageMaker provides a detailed error message, such as:

Received response from MyContainerName1 with status code 200. However, the request payload from MyContainerName1 to MyContainerName2 is 6000000 bytes, which has exceeded the maximum limit of 5 MB.

If a container fails the ping health check while SageMaker is creating an endpoint, it returns a `ClientError` and indicates all of the containers that failed the ping check in the last health check.

# Automatically Scale Amazon SageMaker Models

Amazon SageMaker supports automatic scaling (autoscaling) for your hosted models. *Autoscaling* dynamically adjusts the number of instances provisioned for a model in response to changes in your workload. When the workload increases, autoscaling brings more instances online. When the workload decreases, autoscaling removes unnecessary instances so that you don't pay for provisioned instances that you aren't using.

## Topics

- [Prerequisites \(p. 1931\)](#)
- [Configure model autoscaling with the console \(p. 1934\)](#)
- [Register a model \(p. 1935\)](#)
- [Define a scaling policy \(p. 1936\)](#)
- [Apply a scaling policy \(p. 1938\)](#)
- [Edit a scaling policy \(p. 1939\)](#)
- [Delete a scaling policy \(p. 1940\)](#)
- [Query Endpoint Autoscaling History \(p. 1942\)](#)
- [Update or delete endpoints that use automatic scaling \(p. 1943\)](#)
- [Load testing your autoscaling configuration \(p. 1945\)](#)
- [Use Amazon CloudFormation to update autoscaling policies \(p. 1946\)](#)

## Prerequisites

Before you can use autoscaling, must have already created a Amazon SageMaker model deployment. Deployed models are referred to as a [production variant](#). This includes information about the model and the resources used to host it.

For more information about deploying a model endpoint, see [Deploy the Model to SageMaker Hosting Services \(p. 63\)](#).

To enable autoscaling for a model, you can use the console, the Amazon CLI, or the Application Auto Scaling API. It is recommended to try to [Configure model autoscaling with the console \(p. 1934\)](#) to get familiar with the requirements and to test your first autoscaling configuration. When using the Amazon CLI or Application Auto Scaling the flow is to register the model, define the scaling policy, then apply it. The following overview provides further details on the prerequisites and components used with autoscaling.

## Autoscaling policy overview

To use automatic scaling, you define and apply a scaling policy that uses Amazon CloudWatch metrics and target values that you assign. Automatic scaling uses the policy to increase or decrease the number of instances in response to actual workloads.

You can use the Amazon Web Services Management Console to apply a scaling policy based on a predefined metric. A *predefined metric* is defined in an enumeration so that you can specify it by name in code or use it in the Amazon Web Services Management Console. Alternatively, you can use either the Amazon Command Line Interface (Amazon CLI) or the Application Auto Scaling API to apply a scaling policy based on a predefined or custom metric.

There are two types of supported scaling policies: target-tracking scaling and step scaling. It is recommended to use target-tracking scaling policies for your autoscaling configuration. You configure the *target-tracking scaling policy* by specifying a predefined or custom metric and a target value for the metric. For more information about using Application Auto Scaling target-tracking scaling policies, see [Target Tracking Scaling Policies](#).

You can use step scaling when you require an advanced configuration, such as specifying how many instances to deploy under what conditions. Otherwise, using target-tracking scaling is preferred as it will be fully automated. For more information about using Application Auto Scaling step scaling policies, see [Step Scaling Policies](#).

A scaling policy has the following components:

- A target metric—The Amazon CloudWatch metric that SageMaker automatic scaling uses to determine when and how much to scale.
- Minimum and maximum capacity—The minimum and maximum number of instances to use for scaling.
- A cool down period—The amount of time, in seconds, after a scale-in or scale-out activity completes before another scale-out activity can start.
- Required permissions—Permissions that are required to perform automatic scaling actions.
- A service-linked role—An Amazon Identity and Access Management (IAM) role that is linked to a specific Amazon service. A service-linked role includes all of the permissions that the service requires to call other Amazon services on your behalf. SageMaker automatic scaling automatically generates this role, `AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint`, for you.

## Target metric for autoscaling

Amazon CloudWatch alarms trigger the scaling policy, which calculate how to adjust scaling based on the metric and target value that you set. The scaling policy adds or removes endpoint instances as required to keep the metric at, or close to, the specified target value. In addition, a scaling policy also adjusts to fluctuations in the metric when a workload changes. The scaling policy minimizes rapid fluctuations in the number of available instances for your model.

For example, a scaling policy that uses the predefined `InvocationsPerInstance` metric with a target value of 70 can keep `InvocationsPerInstance` at, or close to 70.

## Minimum and maximum capacity

You may specify the maximum number of endpoint instances for the model. The maximum value must be equal to or greater than the value specified for the minimum number of endpoint instances. SageMaker automatic scaling does not enforce a limit for this value.

You must also specify the minimum number of instances for the model. This value must be at least 1, and equal to or less than the value specified for the maximum number of endpoint instances.

To determine the minimum and maximum number of instances that you need for typical traffic, test your autoscaling configuration with the expected rate of traffic to your model.

### Important

Scaling-in occurs when there is no traffic: if a variant's traffic becomes zero, SageMaker automatically scales in to the minimum number of instances specified. In this case, SageMaker emits metrics with a value of zero. Minimum instance count is required to be 1 or higher.

## Cooldown period

Tune the responsiveness of your scaling policy by adding a cooldown period. A *cooldown period* controls when your model is scaled-in (by reducing instances) or scaled-out (by increasing instances). It does this

by blocking subsequent scale-in or scale-out requests until the period expires. This slows the deletion of instances for scale-in requests, and the creation of instances for scale-out requests. A cooldown period helps to ensure that the scaling policy doesn't launch or terminate additional instances before the previous scaling activity takes effect. After automatic scaling dynamically scales using a scaling policy, it waits for the cooldown period to complete before resuming scaling activities.

You configure the cooldown period in your automatic scaling policy. You can specify the following cooldown periods:

- A scale-in activity reduces the number of instances. A scale-in cooldown period specifies the amount of time, in seconds, after a scale-in activity completes before another scale-in activity can start.
- A scale-out activity increases the number of instances. A scale-out cooldown period specifies the amount of time, in seconds, after a scale-out activity completes before another scale-out activity can start.

If you don't specify a scale-in or a scale-out cooldown period automatic scaling use the default, which is 300 seconds for each.

If instances are being added or removed too quickly when you test your automatic scaling configuration, consider increasing this value. You can see this behavior if the traffic to your model has a lot of spikes, or if you have multiple automatic scaling policies defined for a variant.

If instances are not being added quickly enough to address increased traffic, consider decreasing this value.

## Permissions

The `SagemakerFullAccessPolicy` IAM policy has all of the IAM permissions required to perform autoscaling. For more information about SageMaker IAM permissions, see [SageMaker Roles \(p. 2447\)](#).

If you are using a custom permission policy, you must include the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
        "sagemaker:DescribeEndpoint",
        "sagemaker:DescribeEndpointConfig",
        "sagemaker:UpdateEndpointWeightsAndCapacities"
    ],
    "Resource": "*"
}
{
    "Action": [
        "application-autoscaling:)"
    ],
    "Effect": "Allow",
    "Resource": "*"
}

{
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::*:role/aws-service-role/sagemaker.application-autoscaling.amazonaws.com/
AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint",
    "Condition": {
        "StringLike": { "iam:AWSServiceName": "sagemaker.application-
autoscaling.amazonaws.com" }
    }
}
```

```
        }
      {
        "Effect": "Allow",
        "Action": [
          "cloudwatch:PutMetricAlarm",
          "cloudwatch:DescribeAlarms",
          "cloudwatch:DeleteAlarms"
        ],
        "Resource": "*"
      }
```

## Service-linked role

Autoscaling uses the `AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint` service-linked role; it created for you automatically. A service-linked role is a unique type of IAM role that is linked directly to an Amazon service. Service-linked roles are predefined by the service and include all of the permissions that the service requires to call other Amazon services on your behalf. For more information, see [Service-Linked Roles](#).

## Configure model autoscaling with the console

### To configure autoscaling for a model using the console

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the navigation pane, choose **Endpoints**.
3. Choose the endpoint that you want to configure.
4. For **Endpoint runtime settings**, choose the model variant that you want to configure.
5. For **Endpoint runtime settings**, choose **Configure autoscaling**.

The **Configure variant automatic scaling** page appears.

6. For **Minimum capacity**, type the minimum number of instances that you want the scaling policy to maintain. At least 1 instance is required.
7. For **Maximum capacity**, type the maximum number of instances that you want the scaling policy to maintain.
8. For the **target value**, type the average number of invocations per instance per minute for the model. To determine this value, follow the guidelines in [Load testing \(p. 1945\)](#).

Application Auto Scaling adds or removes instances to keep the metric close to the value that you specify.

9. For **Scale-in cool down (seconds)** and **Scale-out cool down (seconds)**, type the number seconds for each cool down period. Assuming that the order in the list is based on either most important to less important of first applied to last applied.
10. Select **Disable scale in** to prevent the scaling policy from deleting variant instances if you want to ensure that your variant scales out to address increased traffic, but are not concerned with removing instances to reduce costs when traffic decreases, disable scale-in activities.

Scale-out activities are always enabled so that the scaling policy can create endpoint instances as needed.

11. Choose **Save**.

This procedure registers a model as a scalable target with Application Auto Scaling. When you register a model, Application Auto Scaling performs validation checks to ensure the following:

- The model exists

- The permissions are sufficient
- You aren't registering a variant with an instance that is a burstable performance instance such as T2

**Note**

SageMaker doesn't support autoscaling for burstable instances such as T2, because they already allow for increased capacity under increased workloads. For information about burstable performance instances, see [Amazon EC2 Instance Types](#).

## Register a model

You can add autoscaling for a model with the Amazon CLI or the Application Auto Scaling API. You first must register the model, then you must define an autoscaling policy.

### Register a model with the Amazon CLI

With the Amazon CLI, you can configure autoscaling based on either a predefined or a custom metric.

To register your endpoint, use the `register-scalable-target` Amazon CLI command with the following parameters:

- `--service-namespace`—Set this value to `sagemaker`.
- `--resource-id`—The resource identifier for the model (specifically, the production variant). For this parameter, the resource type is `endpoint` and the unique identifier is the name of the production variant. For example, `endpoint/MyEndPoint/variant/MyVariant`.
- `--scalable-dimension`—Set this value to `sagemaker:variant:DesiredInstanceCount`.
- `--min-capacity`—The minimum number of instances that for this model. Set `min-capacity` to at least 1. It must be equal to or less than the value specified for `max-capacity`.
- `--max-capacity`—The maximum number of instances that Application Auto Scaling should manage. Set `max-capacity` to a minimum of 1, It must be equal to or greater than the value specified for `min-capacity`.

#### Example

The following example shows how to register a model named `MyVariant` that is dynamically scaled to have one to eight instances:

```
aws application-autoscaling register-scalable-target \
--service-namespace sagemaker \
--resource-id endpoint/MyEndPoint/variant/MyVariant \
--scalable-dimension sagemaker:variant:DesiredInstanceCount \
--min-capacity 1 \
--max-capacity 8
```

### Register a model with the Application Auto Scaling API

To define the scaling limits for the model, register your model with Application Auto Scaling. Application Auto Scaling dynamically scales the number of production variant instances.

To register your model with Application Auto Scaling, use the `RegisterScalableTarget` Application Auto Scaling API action with the following parameters:

- `ServiceNamespace`—Set this value to `sagemaker`.
- `ResourceId`—The resource identifier for the production variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant, for example `endpoint/MyEndPoint/variant/MyVariant`.

- **ScalableDimension**—Set this value to `sagemaker:variant:DesiredInstanceCount`.
- **MinCapacity**—The minimum number of instances to be managed by Application Auto Scaling. This value must be set to at least 1 and must be equal to or less than the value specified for **MaxCapacity**.
- **MaxCapacity**—The maximum number of instances to be managed by Application Auto Scaling. This value must be set to at least 1 and must be equal to or greater than the value specified for **MinCapacity**.

### Example

The following example shows how to register a SageMaker production variant that is dynamically scaled to use one to eight instances:

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.RegisterScalableTarget
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "ServiceNamespace": "sagemaker",
    "ResourceId": "endpoint/MyEndPoint/variant/MyVariant",
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount",
    "MinCapacity": 1,
    "MaxCapacity": 8
}
```

## Define a scaling policy

To specify the metrics and target values for a scaling policy, you configure a target-tracking scaling policy. You can use either a predefined metric or a custom metric.

Scaling policy configuration is represented by a JSON block. You save your scaling policy configuration as a JSON block in a text file. You use that text file when invoking the Amazon CLI or the Application Auto Scaling API. For more information about policy configuration syntax, see [TargetTrackingScalingPolicyConfiguration](#) in the *Application Auto Scaling API Reference*.

The following options are available for defining a target-tracking scaling policy configuration.

### Use a predefined metric

To quickly define a target-tracking scaling policy for a variant, use the `SageMakerVariantInvocationsPerInstance` predefined metric.

`SageMakerVariantInvocationsPerInstance` is the average number of times per minute that each instance for a variant is invoked. We strongly recommend using this metric.

To use a predefined metric in a scaling policy, create a target tracking configuration for your policy. In the target tracking configuration, include a `PredefinedMetricSpecification` for the predefined metric and a `TargetValue` for the target value of that metric.

### Example

The following example is a typical policy configuration for target-tracking scaling for a variant. In this configuration, we use the `SageMakerVariantInvocationsPerInstance` predefined metric to adjust the number of variant instances so that each instance has a `InvocationsPerInstance` metric of 70.

```
{  
    "TargetValue": 70.0,  
    "PredefinedMetricSpecification":  
    {  
        "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"  
    }  
}
```

## Use a custom metric

If you need to define a target-tracking scaling policy that meets your custom requirements, define a custom metric. You can define a custom metric based on any production variant metric that changes in proportion to scaling.

Not all SageMaker metrics work for target tracking. The metric must be a valid utilization metric, and it must describe how busy an instance is. The value of the metric must increase or decrease in inverse proportion to the number of variant instances. That is, the value of the metric should decrease when the number of instances increases.

**Important**

Before deploying automatic scaling in production, you must test automatic scaling with your custom metric.

### Example

The following example is a target-tracking configuration for a scaling policy. In this configuration, for a variant named `my-variant`, a custom metric adjusts the variant based on an average CPU utilization of 50 percent across all instances.

```
{  
    "TargetValue": 50,  
    "CustomizedMetricSpecification":  
    {  
        "MetricName": "CPUUtilization",  
        "Namespace": "/aws/sagemaker/Endpoints",  
        "Dimensions": [  
            {"Name": "EndpointName", "Value": "my-endpoint"},  
            {"Name": "VariantName", "Value": "my-variant"}  
        ],  
        "Statistic": "Average",  
        "Unit": "Percent"  
    }  
}
```

## Add a cooldown period

To add a cooldown period for scaling-out your model, specify a value, in seconds, for `ScaleOutCooldown`. Similarly, to add a cooldown period for scaling-in your model, add a value, in seconds, for `ScaleInCooldown`. For more information about `ScaleInCooldown` and `ScaleOutCooldown`, see [TargetTrackingScalingPolicyConfiguration](#) in the *Application Auto Scaling API Reference*.

### Example

The following is an example target-tracking configuration for a scaling policy. In this configuration, the `SageMakerVariantInvocationsPerInstance` predefined metric is used to adjust scaling based on an average of 70 across all instances of that variant. The configuration provides a scale-in cooldown period of 10 minutes and a scale-out cooldown period of 5 minutes.

```
{  
    "TargetValue": 70.0,  
    "PredefinedMetricSpecification":  
    {  
        "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"  
    },  
    "ScaleInCooldown": 600,  
    "ScaleOutCooldown": 300  
}
```

## Apply a scaling policy

After registering your model and defining a scaling policy, apply the scaling policy to the registered model. To apply a scaling policy, you can use the Amazon CLI or the Application Auto Scaling API.

### Apply a scaling policy (Amazon CLI)

To apply a scaling policy to your model, use the [put-scaling-policy](#) Amazon CLI command with the following parameters:

- **--policy-name**—The name of the scaling policy.
- **--policy-type**—Set this value to `TargetTrackingScaling`.
- **--resource-id**—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant. For example `endpoint/MyEndpoint/variant/MyVariant`.
- **--service-namespace**—Set this value to `sagemaker`.
- **--scalable-dimension**—Set this value to `sagemaker:variant:DesiredInstanceCount`.
- **--target-tracking-scaling-policy-configuration**—The target-tracking scaling policy configuration to use for the model.

#### Example

The following example uses with Application Auto Scaling to apply a target-tracking scaling policy named `myscalablepolicy` to a model (variant) named `myscalablevariant`. The policy configuration is saved in a file named `config.json`.

```
aws application-autoscaling put-scaling-policy \  
    --policy-name myscalablepolicy \  
    --policy-type TargetTrackingScaling \  
    --resource-id endpoint/MyEndpoint/variant/MyVariant \  
    --service-namespace sagemaker \  
    --scalable-dimension sagemaker:variant:DesiredInstanceCount \  
    --target-tracking-scaling-policy-configuration file://config.json
```

### Apply a scaling policy (Application Auto Scaling API)

To apply a scaling policy to a variant with the Application Auto Scaling API, use the [PutScalingPolicy](#) Application Auto Scaling API action with the following parameters:

- **PolicyName**—The name of the scaling policy.
- **ServiceNamespace**—Set this value to `sagemaker`.
- **ResourceID**—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant. For example, `endpoint/MyEndpoint/variant/MyVariant`.
- **ScalableDimension**—Set this value to `sagemaker:variant:DesiredInstanceCount`.

- **PolicyType**—Set this value to `TargetTrackingScaling`.
- **TargetTrackingScalingPolicyConfiguration**—The target-tracking scaling policy configuration to use for the variant.

### Example

The following example uses Application Auto Scaling to apply a target-tracking scaling policy named `mscalablepolicy` to a variant named `mscalablevariant`. It uses a policy configuration based on the `SageMakerVariantInvocationsPerInstance` predefined metric.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "mscalablepolicy",
    "ServiceNamespace": "sagemaker",
    "ResourceId": "endpoint/MyEndpoint/variant/MyVariant",
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount",
    "PolicyType": "TargetTrackingScaling",
    "TargetTrackingScalingPolicyConfiguration": {
        "TargetValue": 70.0,
        "PredefinedMetricSpecification":
        {
            "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"
        }
    }
}
```

## Edit a scaling policy

You can edit an autoscaling policy with the Amazon Web Services Management Console, the Amazon CLI, or the Application Auto Scaling API.

### Scale-in

Scaling-in occurs when there is no traffic: if a variant's traffic becomes zero, SageMaker automatically scales in to the minimum number of instances specified. In this case, SageMaker emits metrics with a value of zero. Minimum instance count is required to be 1 or higher.

### Disable scale-in activity

You can prevent the target-tracking scaling policy configuration from scaling in your variant by disabling scale-in activity. Disabling scale-in activity prevents the scaling policy from deleting instances, while still allowing it to create them as needed.

To enable or disable scale-in activity for your model, specify a Boolean value for `DisableScaleIn`. For more information about `DisableScaleIn`, see [TargetTrackingScalingPolicyConfiguration](#) in the *Application Auto Scaling API Reference*.

### Example

The following is an example of a target-tracking configuration for a scaling policy where it will scale-out, but not scale-in. In this configuration, the `SageMakerVariantInvocationsPerInstance` predefined

metric will scale-out based on an average of 70 invocations (inference requests) across all instances the model is on. The configuration also disables scale-in activity for the scaling policy.

```
{  
    "TargetValue": 70.0,  
    "PredefinedMetricSpecification":  
    {  
        "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance"  
    },  
    "DisableScaleIn": true  
}
```

## Scale-out

To manually scale-out, adjust the minimum capacity. You can use the console to update this value. Alternatively, use the Amazon CLI with the `--min-capacity` argument, or use the Application Auto Scaling API's `MinCapacity` parameter.

### Disable scale-out activity

To prevent scale-out, adjust the maximum capacity. You can use the console to update this value. Alternatively, use the Amazon CLI with the `--max-capacity` argument, or use the Application Auto Scaling API's `MaxCapacity` parameter.

## Edit a scaling policy (Console)

To edit a scaling policy with the Amazon Web Services Management Console, use the same procedure that you used to [Configure model autoscaling with the console \(p. 1934\)](#).

## Edit a scaling policy (Amazon CLI or Application Auto Scaling API)

You can use the Amazon CLI or the Application Auto Scaling API to edit a scaling policy in the same way that you apply a scaling policy:

- With the Amazon CLI, specify the name of the policy that you want to edit in the `--policy-name` parameter. Specify new values for the parameters that you want to change.
- With the Application Auto Scaling API, specify the name of the policy that you want to edit in the `PolicyName` parameter. Specify new values for the parameters that you want to change.

For more information, see [Apply a scaling policy \(p. 1938\)](#).

## Delete a scaling policy

You can delete a scaling policy with the Amazon Web Services Management Console, the Amazon CLI, or the Application Auto Scaling API. You must delete a scaling policy if you wish to update a model's endpoint.

## Delete a scaling policy (Console)

### To delete an automatic scaling policy (console)

- Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.

2. In the navigation pane, choose **Endpoints**.
3. Choose the endpoint for which you want to delete automatic scaling.
4. For **Endpoint runtime settings**, choose the variant that you want to configure.
5. Choose **Configure auto scaling**.
6. Choose **Deregister auto scaling**.

## Delete a scaling policy (Amazon CLI or Application Auto Scaling API)

You can use the Amazon CLI or the Application Auto Scaling API to delete a scaling policy from a variant.

### Delete a scaling policy (Amazon CLI)

To delete a scaling policy from a variant, use the [delete-scaling-policy](#) Amazon CLI command with the following parameters:

- **--policy-name**—The name of the scaling policy.
- **--resource-id**—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant. For example, `endpoint/MyEndpoint/variant/MyVariant`.
- **--service-namespace**—Set this value to `sagemaker`.
- **--scalable-dimension**—Set this value to `sagemaker:variant:DesiredInstanceCount`.

### Example

The following example deletes a target-tracking scaling policy named `mscalablepolicy` from a variant named `mscalablevariant`.

```
aws application-autoscaling delete-scaling-policy \
  --policy-name mscalablepolicy \
  --resource-id endpoint/MyEndpoint/variant/MyVariant \
  --service-namespace sagemaker \
  --scalable-dimension sagemaker:variant:DesiredInstanceCount
```

### Delete a scaling policy (Application Auto Scaling API)

To delete a scaling policy from your variant, use the [DeleteScalingPolicy](#) Application Auto Scaling API action with the following parameters:

- **PolicyName**—The name of the scaling policy.
- **ServiceNamespace**—Set this value to `sagemaker`.
- **ResourceID**—The resource identifier for the variant. For this parameter, the resource type is `endpoint` and the unique identifier is the name of the variant., For example, `endpoint/MyEndpoint/variant/MyVariant`.
- **ScalableDimension**—Set this value to `sagemaker:variant:DesiredInstanceCount`.

### Example

The following example uses the Application Auto Scaling API to delete a target-tracking scaling policy named `mscalablepolicy` from a variant named `mscalablevariant`.

```
POST / HTTP/1.1
Host: autoscaling.us-east-2.amazonaws.com
Accept-Encoding: identity
X-Amz-Target: AnyScaleFrontendService.DeleteScalingPolicy
X-Amz-Date: 20160506T182145Z
User-Agent: aws-cli/1.10.23 Python/2.7.11 Darwin/15.4.0 botocore/1.4.8
Content-Type: application/x-amz-json-1.1
Authorization: AUTHPARAMS

{
    "PolicyName": "myscalablepolicy",
    "ServiceNamespace": "sagemaker",
    "ResourceId": "endpoint/MyEndpoint/variant/MyVariant",
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount"
}
```

## Query Endpoint Autoscaling History

You can view the status of scaling activities from your endpoint using [DescribeScalingActivities](#). `DescribeScalingActivities` provides descriptive information about the scaling activities in the specified namespace from the previous six weeks.

### How To Query Endpoint Autoscaling Actions

Query your autoscaling endpoints with `DescribeScalingActivities`. To do so, specify `ServiceNameSpace` parameter. `ServiceNameSpace` is the name of the Amazon service that provides the resource.

Valid service name values include the following:

```
ecs | elasticmapreduce | ec2 | appstream | dynamodb | rds | sagemaker | custom-
resource | comprehend | lambda | cassandra
```

In this situation you need to set `ServiceNameSpace` to `sagemaker`.

Use the following Amazon CLI command to view details about all of your `sagemaker` endpoints that have a scaling policy:

```
aws application-autoscaling describe-scaling-activities \
--service-namespace sagemaker
```

You can search for a specific endpoint using `ResourceId`:

```
aws application-autoscaling describe-scaling-activities \
--service-namespace sagemaker \
--resource-id endpoint/<endpoint_name>/variant/<variant_name>
```

When you run this command, it returns the following output:

```
{
    "ActivityId": "activity-id",
    "ServiceNamespace": "sagemaker",
    "ResourceId": "endpoint/<endpoint_name>/variant/<variant_name>",
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount",
    "Description": "string",
    "Cause": "string",
    "StartTime": timestamp,
```

```
        "EndTime": timestamp,  
        "StatusCode": "string",  
        "StatusMessage": "string"  
    }
```

## How to Identify Blocked AutoScaling Due to Instance Quotas

When you scale out or add more instances, you might reach your account-level instance quota. You can use `DescribeScalingActivities` to check whether you have reached your instance quota. When you exceed your quota, automatic scaling is blocked.

To check if you have reached your instance quota, use the Amazon CLI command as shown in the proceeding example where you specified the `ResourceId`:

```
aws application-autoscaling describe-scaling-activities \  
  --service-namespace sagemaker \  
  --resource-id endpoint/<endpoint_name>/variant/<variant_name>
```

Within the return syntax, check the `StatusCode` and `StatusMessage` keys and their associated values. `StatusCode` returns `Failed`. Within `StatusMessage` there is a message indicating that the account-level service quota was reached. The following is an example of what that message might look like:

```
{  
    "ActivityId": "activity-id",  
    "ServiceNamespace": "sagemaker",  
    "ResourceId": "endpoint/<endpoint_name>/variant/<variant_name>",  
    "ScalableDimension": "sagemaker:variant:DesiredInstanceCount",  
    "Description": "string",  
    "Cause": "minimum capacity was set to 110",  
    "StartTime": timestamp,  
    "EndTime": timestamp,  
    "StatusCode": "Failed",  
    "StatusMessage": "Failed to set desired instance count to 110. Reason: The account-level service limit 'ml.xx.xxxxxxx for endpoint usage' is 1000 Instances, with current utilization of 997 Instances and a request delta of 20 Instances. Please contact Amazon support to request an increase for this limit. (Service: AmazonSageMaker; Status Code: 400; Error Code: ResourceLimitExceeded; Request ID: request-id)."  
}
```

## Update or delete endpoints that use automatic scaling

### Topics

- [Update endpoints that use automatic scaling \(p. 1943\)](#)
- [Delete endpoints configured for automatic scaling \(p. 1944\)](#)

## Update endpoints that use automatic scaling

When you update an endpoint, Application Auto Scaling checks to see whether any of the models on that endpoint are targets for automatic scaling. If the update would change the instance type for any model that is a target for automatic scaling, the update fails.

In the Amazon Web Services Management Console, you see a warning that you must deregister the model from automatic scaling before you can update it. If you are trying to update the endpoint by

calling the [UpdateEndpoint](#) API, the call fails. Before you update the endpoint, delete any scaling policies configured for it by calling the [DeleteScalingPolicy](#) Application Auto Scaling API action, then call [DeregisterScalableTarget](#) to deregister the variant as a scalable target. After you update the endpoint, you can register the variant as a scalable target and attach an automatic scaling policy to the updated variant.

There is one exception. If you change the model for a variant that is configured for automatic scaling, Amazon SageMaker automatic scaling allows the update. This is because changing the model doesn't typically affect performance enough to change automatic scaling behavior. If you do update a model for a variant configured for automatic scaling, ensure that the change to the model doesn't significantly affect performance and automatic scaling behavior.

When you update SageMaker endpoints that have automatic scaling applied, complete the following steps:

### To update an endpoint that has automatic scaling applied

1. Deregister the endpoint as a scalable target by calling [DeregisterScalableTarget](#).
2. Because automatic scaling is blocked while the update operation is in progress (or if you turned off automatic scaling in the previous step), you might want to take the additional precaution of increasing the number of instances for your endpoint during the update. To do this, update the instance counts for the production variants hosted at the endpoint by calling [UpdateEndpointWeightsAndCapacities](#).
3. Call [DescribeEndpoint](#) repeatedly until the value of the `EndpointStatus` field of the response is `InService`.
4. Call [DescribeEndpointConfig](#) to get the values of the current endpoint config.
5. Create a new endpoint config by calling [CreateEndpointConfig](#). For the production variants where you want to keep the existing instance count or weight, use the same variant name from the response from the call to [DescribeEndpointConfig](#) in the previous step. For all other values, use the values that you got as the response when you called [DescribeEndpointConfig](#) in the previous step.
6. Update the endpoint by calling [UpdateEndpoint](#). Specify the endpoint config you created in the previous step as the `EndpointConfig` field. If you want to retain the variant properties like instance count or weight, set the value of the `RetainAllVariantProperties` parameter to `True`. This specifies that production variants with the same name will be updated with the most recent `DesiredInstanceCount` from the response from the call to [DescribeEndpoint](#), regardless of the values of the `InitialInstanceCount` field in the new `EndpointConfig`.
7. (Optional) Re-enable automatic scaling by calling [RegisterScalableTarget](#).

#### Note

Steps 1 and 7 are required only if you are updating an endpoint with the following changes:

- Changing the instance type for a production variant that has automatic scaling configured
- Removing a production variant that has automatic scaling configured.

## Delete endpoints configured for automatic scaling

If you delete an endpoint, Application Auto Scaling checks to see whether any of the models on that endpoint are targets for automatic scaling. If any are and you have permission to deregister the model, Application Auto Scaling deregisters those models as scalable targets without notifying you. If you use a custom permission policy that doesn't provide permission for the [DeleteScalingPolicy](#) and [DeregisterScalableTarget](#) actions, you must delete automatic scaling policies and deregister scalable targets and before deleting the endpoint.

**Note**

You, as an IAM user, might not have sufficient permission to delete an endpoint if another IAM user configured automatic scaling for a variant on that endpoint.

## Load testing your autoscaling configuration

Perform load tests to choose an automatic scaling configuration that works the way you want.

For an example of load testing to optimize automatic scaling for an Amazon SageMaker endpoint, see [Load test and optimize an Amazon SageMaker endpoint using automatic scaling](#).

The following guidelines for load testing assume you are using an automatic scaling policy that uses the predefined target metric `SageMakerVariantInvocationsPerInstance`.

**Topics**

- [Determine the performance characteristics \(p. 1945\)](#)
- [Calculate the target load \(p. 1945\)](#)

## Determine the performance characteristics

Perform load testing to find the peak `InvocationsPerInstance` that your model's production variant can handle, and the latency of requests, as concurrency increases.

This value depends on the instance type chosen, payloads that clients of your model typically send, and the performance of any external dependencies your model has.

### To find the peak requests-per-second (RPS) your model's production variant can handle and latency of requests

1. Set up an endpoint with your model using a single instance. For information about how to set up an endpoint, see [Deploy the Model to SageMaker Hosting Services \(p. 63\)](#).
2. Use a load testing tool to generate an increasing number of parallel requests, and monitor the RPS and model latency in the output of the load testing tool.

**Note**

You can also monitor requests-per-minute instead of RPS. In that case don't multiply by 60 in the equation to calculate `SageMakerVariantInvocationsPerInstance` shown below.

When the model latency increases or the proportion of successful transactions decreases, this is the peak RPS that your model can handle.

## Calculate the target load

After you find the performance characteristics of the variant, you can determine the maximum RPS we should allow to be sent to an instance. The threshold used for scaling must be less than this maximum value. Use the following equation in combination with load testing to determine the correct value for the `SageMakerVariantInvocationsPerInstance` target metric in your automatic scaling configuration.

```
SageMakerVariantInvocationsPerInstance = (MAX_RPS * SAFETY_FACTOR) * 60
```

Where `MAX_RPS` is the maximum RPS that you determined previously, and `SAFETY_FACTOR` is the safety factor that you chose to ensure that your clients don't exceed the maximum RPS. Multiply by 60 to convert from RPS to invocations-per-minute to match the per-minute CloudWatch metric that SageMaker uses to implement automatic scaling (you don't need to do this if you measured requests-per-minute instead of requests-per-second).

**Note**

SageMaker recommends that you start testing with a `SAFETY_FACTOR` of 0.5. Test your automatic scaling configuration to ensure it operates in the way you expect with your model for both increasing and decreasing customer traffic on your endpoint.

## Use Amazon CloudFormation to update autoscaling policies

The following is an example for how to enable autoscaling on an endpoint using Amazon CloudFormation.

```
Endpoint:  
  Type: "AWS::SageMaker::Endpoint"  
  Properties:  
    EndpointName: yourEndpointName  
    EndpointConfigName: yourEndpointConfigName  
  
  ScalingTarget:  
    Type: "AWS::ApplicationAutoScaling::ScalableTarget"  
    Properties:  
      MaxCapacity: 10  
      MinCapacity: 2  
      ResourceId: endpoint/MyEndPoint/variant/MyVariant  
      RoleARN: arn  
      ScalableDimension: sagemaker:variant:DesiredInstanceCount  
      ServiceNamespace: sagemaker  
  
  ScalingPolicy:  
    Type: "AWS::ApplicationAutoScaling::ScalingPolicy"  
    Properties:  
      PolicyName: myscalablepolicy  
      PolicyType: TargetTrackingScaling  
      ScalingTargetId:  
        Ref: ScalingTarget  
      TargetTrackingScalingPolicyConfiguration:  
        TargetValue: 75.0  
        ScaleInCooldown: 600  
        ScaleOutCooldown: 30  
      PredefinedMetricSpecification:  
        PredefinedMetricType: SageMakerVariantInvocationsPerInstance
```

For more details, refer to CloudFront's [AutoScalingPlans API reference](#).

## Host Instance Storage Volumes

When you create an endpoint, Amazon SageMaker attaches an Amazon Elastic Block Store (Amazon EBS) storage volume to Amazon EC2 instances that hosts the endpoint. The size of the storage volume is scalable, and storage options are divided into two categories: SSD-backed storage and HDD-backed storage.

For more information about Amazon EBS storages and features, see the following pages.

- [Amazon EBS Features](#)
- [Amazon EBS User Guide](#)

For a full list of the host instance storage volumes, see [Host Instance Storage Volumes Table](#)

# Test models in production

In production ML workflows, data scientists and engineers frequently try to improve their models in various ways, such as by performing [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#), training on additional or more-recent data, and improving feature selection. Performing A/B testing between a new model and an old model with production traffic can be an effective final step in the validation process for a new model. In A/B testing, you test different variants of your models and compare how each variant performs. If the newer version of the model delivers better performance than the previously-existing version, replace the old version of the model with the new version in production.

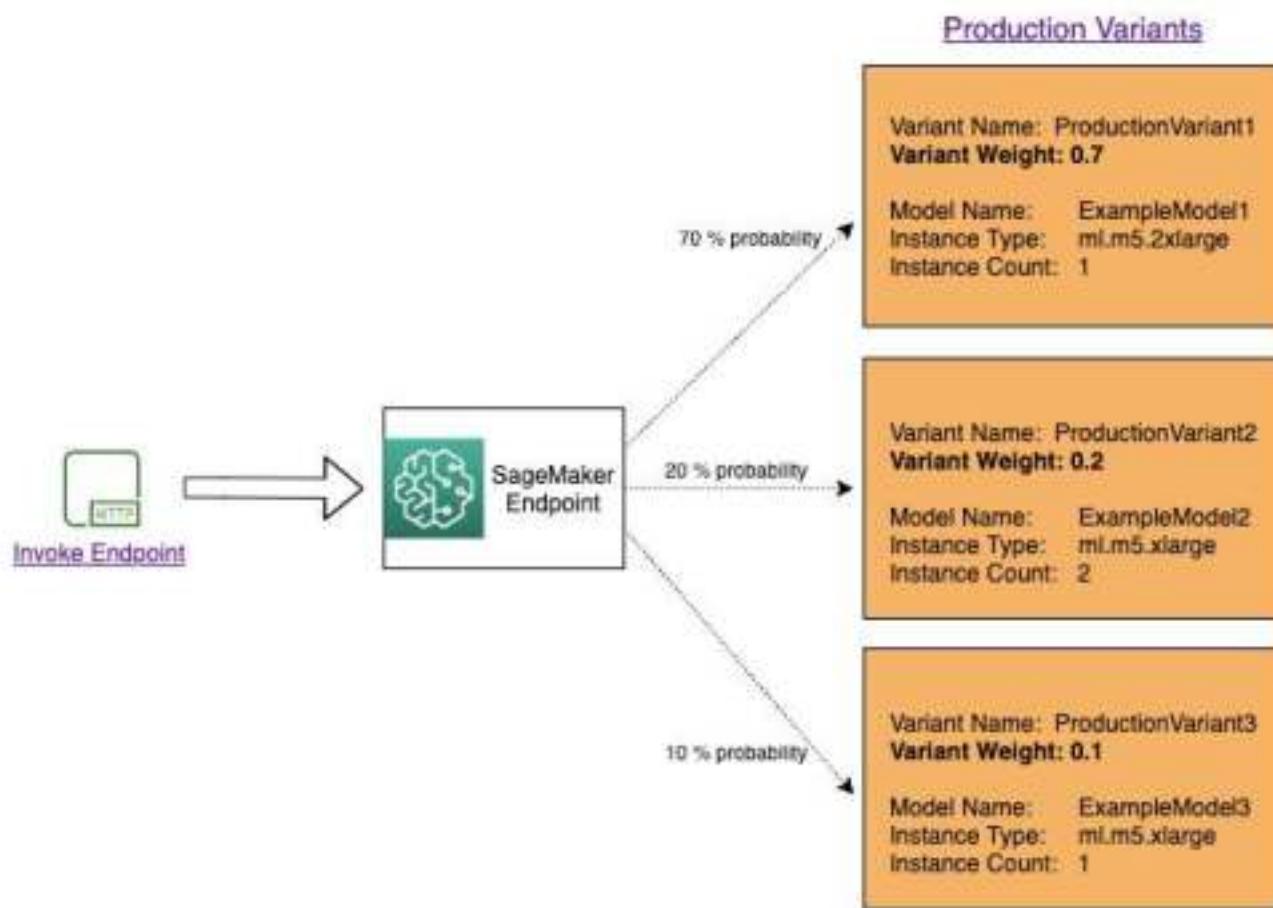
Amazon SageMaker enables you to test multiple models or model versions behind the same endpoint using production variants. Each production variant identifies a machine learning (ML) model and the resources deployed for hosting the model. By using production variants, you can test ML models that have been trained using different datasets, trained using different algorithms and ML frameworks, or are deployed to different instance type, or any combination of all of these. You can distribute endpoint invocation requests across multiple production variants by providing the traffic distribution for each variant, or you can invoke a specific variant directly for each request. In this topic, we look at both methods for testing ML models.

## Topics

- [Test models by specifying traffic distribution \(p. 1947\)](#)
- [Test models by invoking specific variants \(p. 1948\)](#)
- [Model A/B test example \(p. 1949\)](#)

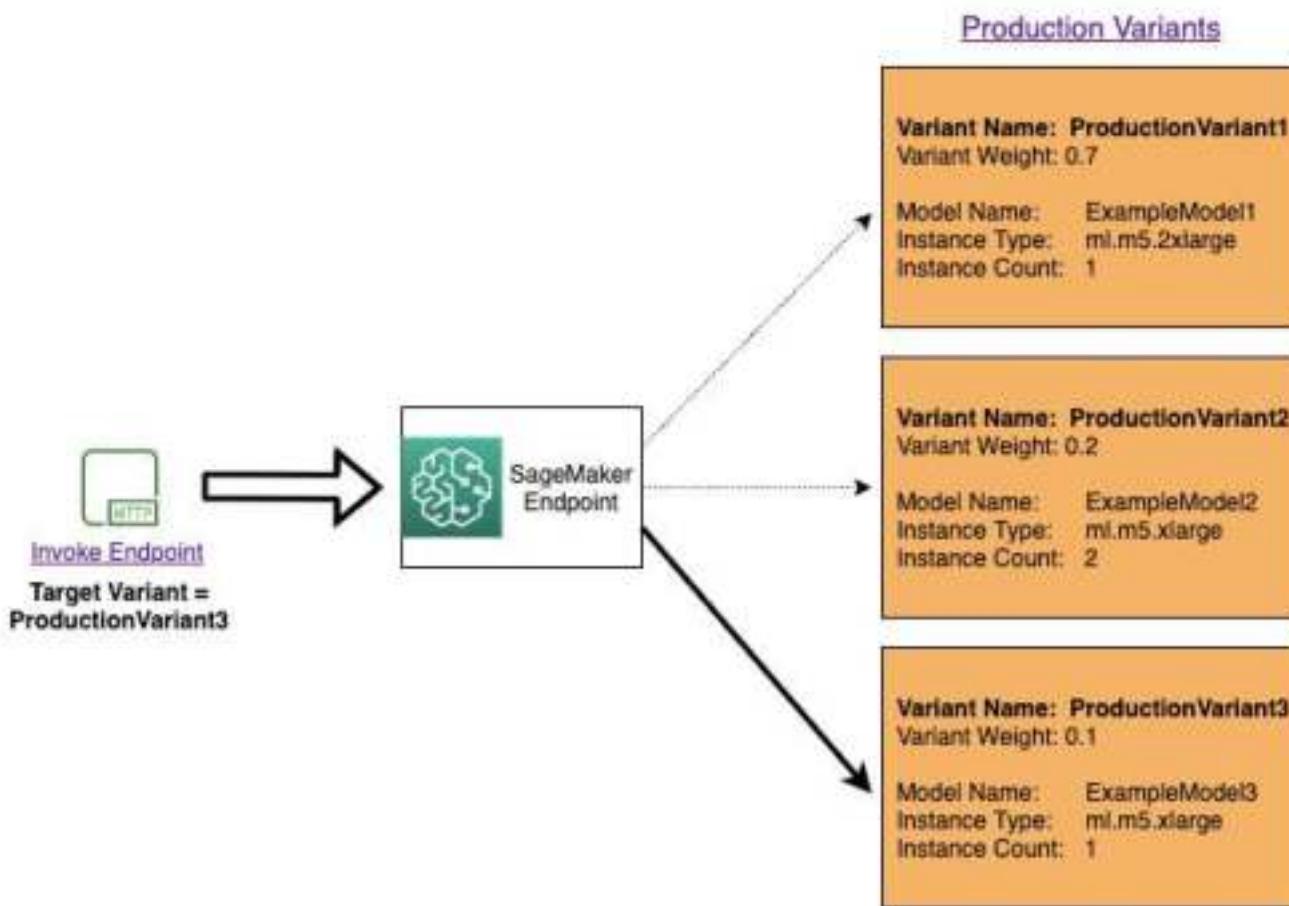
## Test models by specifying traffic distribution

To test multiple models by distributing traffic between them, specify the percentage of the traffic that gets routed to each model by specifying the weight for each production variant in the endpoint configuration. For information, see [CreateEndpointConfig](#). The following diagram shows how this works in more detail.



## Test models by invoking specific variants

To test multiple models by invoking specific models for each request, specify the specific version of the model you want to invoke by providing a value for the `TargetVariant` parameter when you call [InvokeEndpoint](#). SageMaker ensures that the request is processed by the production variant you specify. If you have already provided traffic distribution and specify a value for the `TargetVariant` parameter, the targeted routing overrides the random traffic distribution. The following diagram shows how this works in more detail.



## Model A/B test example

The following example shows how to perform A/B model testing. For a sample notebook that implements this example, see ["A/B Testing ML models in production."](#)

### Step 1: Create and deploy models

First, we define where our models are located in Amazon S3. These locations are used when we deploy our models in subsequent steps:

```
model_url = f"s3://{path_to_model_1}"
model_url2 = f"s3://{path_to_model_2}"
```

Next, we create the model objects with the image and model data. These model objects are used to deploy production variants on an endpoint. The models are developed by training ML models on different data sets, different algorithms or ML frameworks, and different hyperparameters:

```
from sagemaker.amazon.amazon_estimator import get_image_uri
model_name = f"DEMO-xgb-churn-pred-{datetime.now():%Y-%m-%d-%H-%M-%S}"
model_name2 = f"DEMO-xgb-churn-pred2-{datetime.now():%Y-%m-%d-%H-%M-%S}"
```

```

image_uri = get_image_uri(boto3.Session().region_name, 'xgboost', '0.90-1')
image_uri2 = get_image_uri(boto3.Session().region_name, 'xgboost', '0.90-2')

sm_session.create_model(name=model_name, role=role, container_defs={
    'Image': image_uri,
    'ModelDataUrl': model_url
})

sm_session.create_model(name=model_name2, role=role, container_defs={
    'Image': image_uri2,
    'ModelDataUrl': model_url2
})

```

We now create two production variants, each with its own different model and resource requirements (instance type and counts). This enables you to also test models on different instance types.

We set an initial\_weight of 1 for both variants. This means that 50% of requests go to Variant1, and the remaining 50% of requests to Variant2. The sum of weights across both variants is 2 and each variant has weight assignment of 1. This means that each variant receives 1/2, or 50%, of the total traffic.

```

from sagemaker.session import production_variant

variant1 = production_variant(model_name=model_name,
                               instance_type="ml.m5.xlarge",
                               initial_instance_count=1,
                               variant_name='Variant1',
                               initial_weight=1)

variant2 = production_variant(model_name=model_name2,
                               instance_type="ml.m5.xlarge",
                               initial_instance_count=1,
                               variant_name='Variant2',
                               initial_weight=1)

```

Finally we're ready to deploy these production variants on a SageMaker endpoint.

```

endpoint_name = f"DEMO-xgb-churn-pred-{datetime.now():%Y-%m-%d-%H-%M-%S}"
print(f"EndpointName={endpoint_name}")

sm_session.endpoint_from_production_variants(
    name=endpoint_name,
    production_variants=[variant1, variant2]
)

```

## Step 2: Invoke the deployed models

Now we send requests to this endpoint to get inferences in real time. We use both traffic distribution and direct targeting.

First, we use traffic distribution that we configured in the previous step. Each inference response contains the name of the production variant that processes the request, so we can see that traffic to the two production variants is roughly equal.

```

# get a subset of test data for a quick test
!tail -120 test_data/test-dataset-input-cols.csv > test_data/
test_sample_tail_input_cols.csv
print(f"Sending test traffic to the endpoint {endpoint_name}. \nPlease wait...")

```

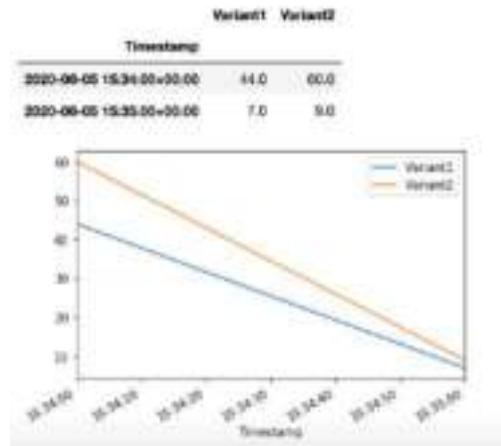
```

with open('test_data/test_sample_tail_input_cols.csv', 'r') as f:
    for row in f:
        print(".", end="", flush=True)
        payload = row.rstrip('\n')
        sm_runtime.invoke_endpoint(EndpointName=endpoint_name,
                                    ContentType="text/csv",
                                    Body=payload)
        time.sleep(0.5)

print("Done!")

```

SageMaker emits metrics such as Latency and Invocations for each variant in Amazon CloudWatch. For a complete list of metrics that SageMaker emits, see [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 2523\)](#). Let's query CloudWatch to get the number of invocations per variant, to show how invocations are split across variants by default:



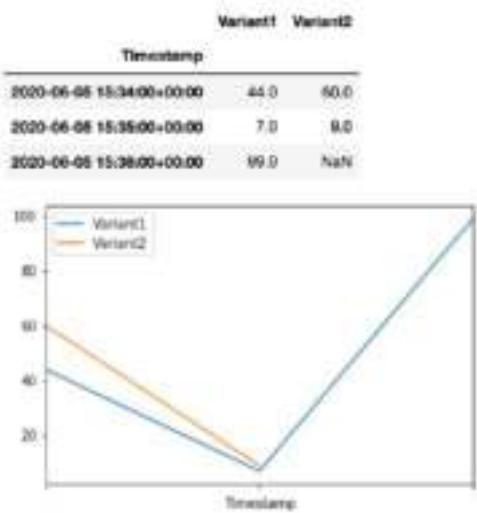
Now let's invoke a specific version of the model by specifying `Variant1` as the `TargetVariant` in the call to `invoke_endpoint`.

```

print(f"Sending test traffic to the endpoint {endpoint_name}. \nPlease wait...")
with open('test_data/test_sample_tail_input_cols.csv', 'r') as f:
    for row in f:
        print(".", end="", flush=True)
        payload = row.rstrip('\n')
        sm_runtime.invoke_endpoint(EndpointName=endpoint_name,
                                    ContentType="text/csv",
                                    Body=payload,
                                    *TargetVariant="Variant1") # Notice the new parameter
        time.sleep(0.5)

```

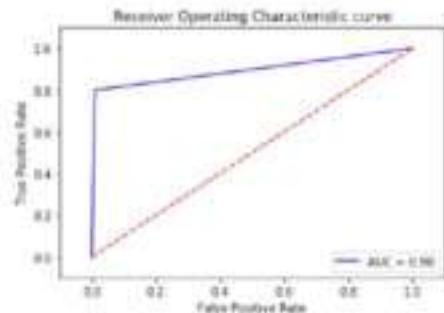
To confirm that all new invocations were processed by `Variant1`, we can query CloudWatch to get the number of invocations per variant. We see that for the most recent invocations (latest timestamp), all requests were processed by `Variant1`, as we had specified. There were no invocations made for `Variant2`.



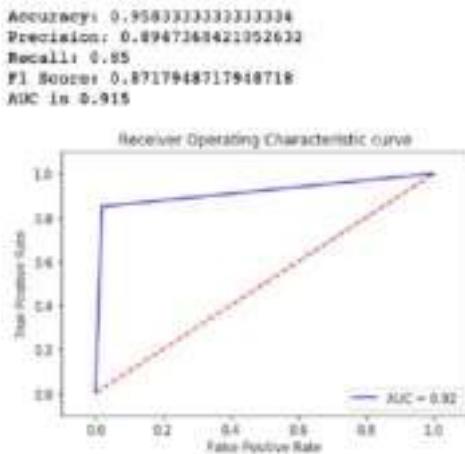
## Step 3: Evaluate model performance

To see which model version performs better, let's evaluate the accuracy, precision, recall, F1 score, and Receiver operating characteristic/Area under the curve for each variant. First, let's look at these metrics for Variant1:

```
Accuracy: 0.9533333333333334
Precision: 0.8111764705882751
Recall: 0.8
F1 Score: 0.8666666666666667
AUC is 0.895
```



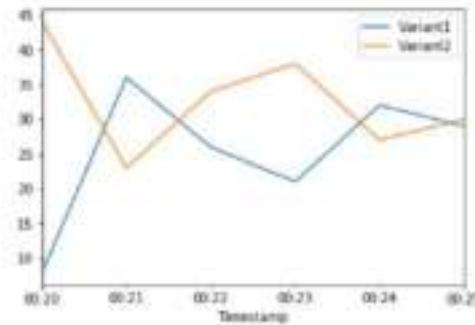
Now let's look at the metrics for Variant2:



For most of our defined metrics, Variant2 is performing better, so this is the one that we want to use in production.

## Step 4: Increase traffic to the best model

Now that we have determined that Variant2 performs better than Variant1, we shift more traffic to it. We can continue to use `TargetVariant` to invoke a specific model variant, but a simpler approach is to update the weights assigned to each variant by calling [UpdateEndpointWeightsAndCapacities](#). This changes the traffic distribution to your production variants without requiring updates to your endpoint. Recall from the setup section that we set variant weights to split traffic 50/50. The CloudWatch metrics for the total invocations for each variant below show us the invocation patterns for each variant:



Now we shift 75% of the traffic to Variant2 by assigning new weights to each variant using `UpdateEndpointWeightsAndCapacities`. SageMaker now sends 75% of the inference requests to Variant2 and remaining 25% of requests to Variant1.

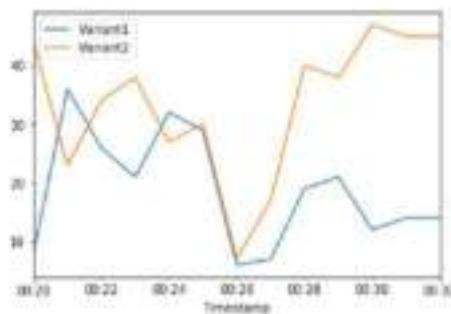
```

sm.update_endpoint_weights_and_capacities(
    EndpointName=endpoint_name,
    DesiredWeightsAndCapacities=[
        {
            "DesiredWeight": 25,
            "VariantName": variant1["VariantName"]
        },
        {
            "DesiredWeight": 75,
            "VariantName": variant2["VariantName"]
        }
    ]
)

```

)

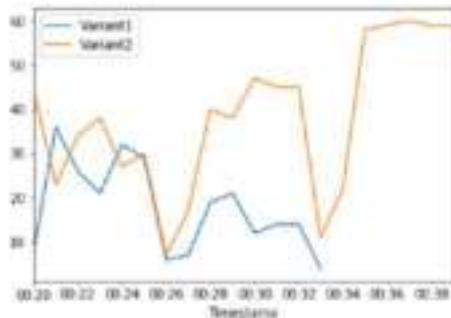
The CloudWatch metrics for total invocations for each variant shows us higher invocations for Variant2 than for Variant1:



We can continue to monitor our metrics, and when we're satisfied with a variant's performance, we can route 100% of the traffic to that variant. We use `UpdateEndpointWeightsAndCapacities` to update the traffic assignments for the variants. The weight for Variant1 is set to 0 and the weight for Variant2 is set to 1. SageMaker now sends 100% of all inference requests to Variant2.

```
sm.update_endpoint_weights_and_capacities(  
    EndpointName=endpoint_name,  
    DesiredWeightsAndCapacities=[  
        {  
            "DesiredWeight": 0,  
            "VariantName": variant1["VariantName"]  
        },  
        {  
            "DesiredWeight": 1,  
            "VariantName": variant2["VariantName"]  
        }  
    ]  
)
```

The CloudWatch metrics for the total invocations for each variant show that all inference requests are being processed by Variant2 and there are no inference requests processed by Variant1.



You can now safely update your endpoint and delete Variant1 from your endpoint. You can also continue testing new models in production by adding new variants to your endpoint and following steps 2 - 4.

# Troubleshoot Amazon SageMaker Model Deployments

If you encounter an issue when deploying machine learning models in Amazon SageMaker, see the following guidance.

## Topics

- [Detection Errors in the Active CPU Count \(p. 1955\)](#)

## Detection Errors in the Active CPU Count

If you deploy a SageMaker model with a Linux Java Virtual Machine (JVM), you might encounter detection errors that prevent using available CPU resources. This issue affects some JVMs that support Java 8 and Java 9, and most that support Java 10 and Java 11. These JVMs implement a mechanism that detects and handles the CPU count and the maximum memory available when running a model in a Docker container, and, more generally, within Linux `taskset` commands or control groups (`cgroups`). SageMaker deployments take advantage of some of the settings that the JVM uses for managing these resources. Currently, this causes the container to incorrectly detect the number of available CPUs.

SageMaker doesn't limit access to CPUs on an instance. However, the JVM might detect the CPU count as 1 when more CPUs are available for the container. As a result, the JVM adjusts all of its internal settings to run as if only 1 CPU core is available. These settings affect garbage collection, locks, compiler threads, and other JVM internals that negatively affect the concurrency, throughput, and latency of the container.

For an example of the misdetection, in a container configured for SageMaker that is deployed with a JVM that is based on Java8\_191 and that has four available CPUs on the instance, run the following command to start your JVM:

```
java -XX:+UnlockDiagnosticVMOptions -XX:+PrintActiveCpus -version
```

This generates the following output:

```
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: determined by OSContainer: 1
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: determined by OSContainer: 1
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: determined by OSContainer: 1
active_processor_count: sched_getaffinity processor count: 4
active_processor_count: determined by OSContainer: 1
openjdk version "1.8.0_191"
OpenJDK Runtime Environment (build 1.8.0_191-8u191-b12-2ubuntu0.16.04.1-b12)
OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)
```

Many of the JVMs affected by this issue have an option to disable this behavior and reestablish full access to all of the CPUs on the instance. Disable the unwanted behavior and establish full access to all instance CPUs by including the `-XX:-UseContainerSupport` parameter when starting Java applications. For example, run the `java` command to start your JVM as follows:

```
java -XX:-UseContainerSupport -XX:+UnlockDiagnosticVMOptions -XX:+PrintActiveCpus -version
```

This generates the following output:

```
active_processor_count: sched_getaffinity processor count: 4
openjdk version "1.8.0_191"
OpenJDK Runtime Environment (build 1.8.0_191-8u191-b12-2ubuntu0.16.04.1-b12)
OpenJDK 64-Bit Server VM (build 25.191-b12, mixed mode)
```

Check whether the JVM used in your container supports the `-XX:-UseContainerSupport` parameter. If it does, always pass the parameter when you start your JVM. This provides access to all of the CPUs in your instances.

You might also encounter this issue when indirectly using a JVM in SageMaker containers. For example, when using a JVM to support SparkML Scala. The `-XX:-UseContainerSupport` parameter also affects the output returned by the `Java Runtime.getRuntime().availableProcessors()` API .

## Deployment Best Practices

This topic provides guidance on best practices for deploying machine learning models in Amazon SageMaker.

### Deploy Multiple Instances Across Availability Zones

**Create robust endpoints when hosting your model.** SageMaker endpoints can help protect your application from [Availability Zone](#) outages and instance failures. If an outage occurs or an instance fails, SageMaker automatically attempts to distribute your instances across Availability Zones. For this reason, we strongly recommended that you deploy multiple instances for each production endpoint.

If you are using an [Amazon Virtual Private Cloud \(VPC\)](#), configure the VPC with at least two [Subnets](#), each in a different Availability Zone. If an outage occurs or an instance fails, Amazon SageMaker automatically attempts to distribute your instances across Availability Zones.

In general, to achieve more reliable performance, use more small [Instance Types](#) in different Availability Zones to host your endpoints.

## Amazon SageMaker Model Monitor

Amazon SageMaker Model Monitor continuously monitors the quality of Amazon SageMaker machine learning models in production. With Model Monitor, you can set alerts that notify you when there are deviations in the model quality. Early and proactive detection of these deviations enables you to take corrective actions, such as retraining models, auditing upstream systems, or fixing quality issues without having to monitor models manually or build additional tooling. You can use Model Monitor prebuilt monitoring capabilities that do not require coding. You also have the flexibility to monitor models by coding to provide custom analysis.

Model Monitor provides the following types of monitoring:

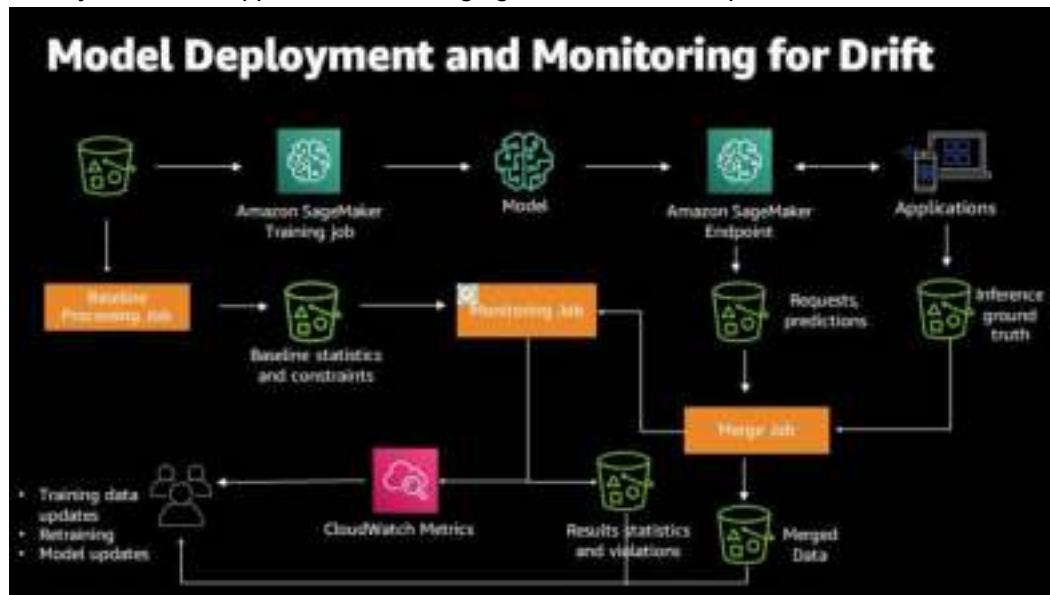
- [Monitor Data Quality \(p. 1958\)](#) - Monitor drift in data quality.
- [Monitor Model Quality \(p. 1963\)](#) - Monitor drift in model quality metrics, such as accuracy.
- [Monitor Bias Drift for Models in Production \(p. 1970\)](#) - Monitor bias in your model's predictions.
- [Monitor Feature Attribution Drift for Models in Production \(p. 1973\)](#) - Monitor drift in feature attribution.

## Topics

- [How Model Monitor Works \(p. 1957\)](#)
- [Monitor Data Quality \(p. 1958\)](#)
- [Monitor Model Quality \(p. 1963\)](#)
- [Monitor Bias Drift for Models in Production \(p. 1970\)](#)
- [Monitor Feature Attribution Drift for Models in Production \(p. 1973\)](#)
- [Capture Data \(p. 1977\)](#)
- [Schedule Monitoring Jobs \(p. 1979\)](#)
- [Amazon SageMaker Model Monitor Prebuilt Container \(p. 1981\)](#)
- [Interpret Results \(p. 1982\)](#)
- [Visualize Results in Amazon SageMaker Studio \(p. 1984\)](#)
- [Advanced Topics \(p. 1990\)](#)

## How Model Monitor Works

Amazon SageMaker Model Monitor automatically monitors machine learning (ML) models in production and notifies you when quality issues arise. Model Monitor uses rules to detect drift in your models and alerts you when it happens. The following figure shows how this process works.



To enable model monitoring, you take the following steps, which follow the path of the data through the various data collection, monitoring, and analysis processes:

- Enable the endpoint to capture data from incoming requests to a trained ML model and the resulting model predictions.
- Create a baseline from the dataset that was used to train the model. The baseline computes metrics and suggests constraints for the metrics. Real-time predictions from your model are compared to the constraints, and are reported as violations if they are outside the constrained values.
- Create a monitoring schedule specifying what data to collect, how often to collect it, how to analyze it, and which reports to produce.
- Inspect the reports, which compare the latest data with the baseline, and watch for any violations reported and for metrics and notifications from Amazon CloudWatch.

## Notes

- Model Monitor currently supports only tabular data.
- Model Monitor currently supports only endpoints that host a single model and does not support monitoring multi-model endpoints. For information on using multi-model endpoints, see [Host Multiple Models with Multi-Model Endpoints \(p. 1895\)](#).
- Model Monitor supports monitoring inference pipelines, but capturing and analyzing data is done for the entire pipeline, not for individual containers in the pipeline.
- To prevent impact to inference requests, Data Capture stops capturing requests at high levels of disk usage. It is recommended you keep your disk utilization below 75% in order to ensure data capture continues capturing requests.
- If you launch SageMaker Studio in a custom Amazon VPC, you need to create VPC endpoints to enable Model Monitor to communicate with Amazon S3 and CloudWatch. For information about VPC endpoints, see [VPC endpoints](#) in the *Amazon Virtual Private Cloud User Guide*. For information about launching SageMaker Studio in a custom VPC, see [Connect SageMaker Studio Notebooks to Resources in a VPC \(p. 2491\)](#).

## Model Monitor Sample Notebooks

For a sample notebook that takes you through the full end-to-end workflow for Model Monitor, see the [Introduction to Amazon SageMaker Model Monitor](#).

For a sample notebook that visualizes the statistics.json file for a selected execution in a monitoring schedule, see the [Model Monitor Visualization](#).

For instructions that show you how to create and access Jupyter notebook instances that you can use to run the example in SageMaker, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). After you have created a notebook instance and opened it, choose the **SageMaker Examples** tab to see a list of all the SageMaker samples. To open a notebook, choose the notebook's **Use** tab and choose **Create copy**.

## Monitor Data Quality

Data quality monitoring automatically monitors machine learning (ML) models in production and notifies you when data quality issues arise. ML models in production have to make predictions on real-life data that is not carefully curated like most training datasets. If the statistical nature of the data that your model receives while in production drifts away from the nature of the baseline data it was trained on, the model begins to lose accuracy in its predictions. Amazon SageMaker Model Monitor uses rules to detect data drift and alerts you when it happens. To monitor data quality, follow these steps:

- Enable data capture. This captures inference input and output from a real-time inference endpoint and stores the data in Amazon S3. For more information, see [Capture Data \(p. 1977\)](#).
- Create a baseline. In this step, you run a baseline job that analyzes an input dataset that you provide. The baseline computes baseline schema constraints and statistics for each feature using [Deequ](#), an open source library built on Apache Spark, which is used to measure data quality in large datasets. For more information, see [Create a Baseline \(p. 1959\)](#).
- Define and schedule data quality monitoring jobs. For more information, see [Schedule Monitoring Jobs \(p. 1979\)](#).
- View data quality metrics. For more information, see [Schema for Statistics \(statistics.json file\) \(p. 1960\)](#).
- Integrate data quality monitoring with Amazon CloudWatch. For more information, see [CloudWatch Metrics \(p. 1961\)](#).
- Interpret the results of a monitoring job. For more information, see [Interpret Results \(p. 1982\)](#).
- Use SageMaker Studio to enable data quality monitoring and visualize results. For more information, see [Visualize Results in Amazon SageMaker Studio \(p. 1984\)](#).

**Note**

Amazon SageMaker Model Monitor currently supports only tabular data.

**Topics**

- [Create a Baseline \(p. 1959\)](#)
- [Schema for Statistics \(statistics.json file\) \(p. 1960\)](#)
- [CloudWatch Metrics \(p. 1961\)](#)
- [Schema for Violations \(constraint\\_violations.json file\) \(p. 1962\)](#)

## Create a Baseline

The baseline calculations of statistics and constraints are needed as a standard against which data drift and other data quality issues can be detected. Model Monitor provides a built-in container that provides the ability to suggest the constraints automatically for CSV and flat JSON input. This *sagemaker-model-monitor-analyzer* container also provides you with a range of model monitoring capabilities, including constraint validation against a baseline, and emitting Amazon CloudWatch metrics. This container is based on Spark and is built with [Deequ](#). All column names in your baseline dataset must be compliant with Spark. For column names, use only lowercase characters, and `_` as the only special character.

The training dataset that you used to trained the model is usually a good baseline dataset. The training dataset data schema and the inference dataset schema should exactly match (the number and order of the features). Note that the prediction/output columns are assumed to be the first columns in the training dataset. From the training dataset, you can ask SageMaker to suggest a set of baseline constraints and generate descriptive statistics to explore the data. For this example, upload the training dataset that was used to train the pretrained model included in this example. If you already stored the training dataset in Amazon S3, you can point to it directly.

### To Create a baseline from a training dataset

When you have your training data ready and stored in Amazon S3, start a baseline processing job with `DefaultModelMonitor.suggest_baseline()` using the [Amazon SageMaker Python SDK](#). This uses an [Amazon SageMaker Model Monitor Prebuilt Container \(p. 1981\)](#) that generates baseline statistics and suggests baseline constraints for the dataset and writes them to the `output_s3_uri` location that you specify.

```
from sagemaker.model_monitor import DefaultModelMonitor
from sagemaker.model_monitor.dataset_format import DatasetFormat

my_default_monitor = DefaultModelMonitor(
    role=role,
    instance_count=1,
    instance_type='ml.m5.xlarge',
    volume_size_in_gb=20,
    max_runtime_in_seconds=3600,
)

my_default_monitor.suggest_baseline(
    baseline_dataset=baseline_data_uri+'/training-dataset-with-header.csv',
    dataset_format=DatasetFormat.csv(header=True),
    output_s3_uri=baseline_results_uri,
    wait=True
)
```

**Note**

If you provide the feature/column names in the training dataset as the first row and set the `header=True` option as shown in the previous code sample, SageMaker uses the feature name in the constraints and statistics file.

The baseline statistics for the dataset are contained in the `statistics.json` file and the suggested baseline constraints are contained in the `constraints.json` file in the location you specify with `output_s3_uri`.

### Output Files for Tabular Dataset Statistics and Constraints

File Name	Description
<code>statistics.json</code>	This file is expected to have columnar statistics for each feature in the dataset that is analyzed. For more information about the schema for this file, see <a href="#">Schema for Statistics (statistics.json file) (p. 1995)</a> .
<code>constraints.json</code>	This file is expected to have the constraints on the features observed. For more information about the schema for this file, see <a href="#">Schema for Constraints (constraints.json file) (p. 1997)</a> .

The [Amazon SageMaker Python SDK](#) provides convenience functions described to generate the baseline statistics and constraints. But if you want to call processing job directly for this purpose instead, you need to set the `Environment` map as shown in the following example:

```
"Environment": {
    "dataset_format": "{\"csv\": { \"header\": true}}",
    "dataset_source": "/opt/ml/processing/sm_input",
    "output_path": "/opt/ml/processing/sm_output",
    "publish_cloudwatch_metrics": "Disabled",
}
```

### Schema for Statistics (statistics.json file)

Amazon SageMaker Model Monitor prebuilt container computes per column/feature statistics. The statistics are calculated for the baseline dataset and also for the current dataset that is being analyzed.

```
{
    "version": 0,
    # dataset level stats
    "dataset": {
        "item_count": number
    },
    # feature level stats
    "features": [
        {
            "name": "feature-name",
            "inferred_type": "Fractional" | "Integral",
            "numerical_statistics": {
                "common": {
                    "num_present": number,
                    "num_missing": number
                },
                "mean": number,
                "sum": number,
                "std_dev": number,
                "min": number,
                "max": number,
                "distribution": {
                    "kll": {
                        "buckets": [
                            {
                                "lower_bound": number,

```

```

        "upper_bound": number,
        "count": number
    }
],
"sketch": {
    "parameters": {
        "c": number,
        "k": number
    },
    "data": [
        [
            [
                num,
                num,
                num,
                num
            ],
            [
                num,
                num
            ][
                num,
                num
            ]
        ]
    }
} #sketch
} #KLL
} #distribution
} #num_stats
},
{
    "name": "feature-name",
    "inferred_type": "String",
    "string_statistics": {
        "common": {
            "num_present": number,
            "num_missing": number
        },
        "distinct_count": number,
        "distribution": {
            "categorical": {
                "buckets": [
                    {
                        "value": "string",
                        "count": number
                    }
                ]
            }
        }
    },
    "#provision for custom stats"
}
]
}
}

```

Note the following:

- The prebuilt containers compute [KLL sketch](#), which is a compact quantiles sketch.
- By default, we materialize the distribution in 10 buckets. This is not currently configurable.

## CloudWatch Metrics

You can use the built-in Amazon SageMaker Model Monitor container for CloudWatch metrics. When the `emit_metrics` option is `Enabled` in the baseline constraints file, SageMaker emits these metrics

for each feature/column observed in the dataset in the `/aws/sagemaker/Endpoints/data-metric` namespace with `EndpointName` and `ScheduleName` dimensions.

For numerical fields, the built-in container emits the following CloudWatch metrics:

- Metric: Max → query for `MetricName: feature_data_{feature_name}`, Stat: Max
- Metric: Min → query for `MetricName: feature_data_{feature_name}`, Stat: Min
- Metric: Sum → query for `MetricName: feature_data_{feature_name}`, Stat: Sum
- Metric: SampleCount → query for `MetricName: feature_data_{feature_name}`, Stat: SampleCount
- Metric: Average → query for `MetricName: feature_data_{feature_name}`, Stat: Average

For both numerical and string fields, the built-in container emits the following CloudWatch metrics:

- Metric: Completeness → query for `MetricName: feature_non_null_{feature_name}`, Stat: Sum
- Metric: Baseline Drift → query for `MetricName: feature_baseline_drift_{feature_name}`, Stat: Sum

## Schema for Violations (constraint\_violations.json file)

The violations file is generated as the output of a `MonitoringExecution`, which lists the results of evaluating the constraints (specified in the `constraints.json` file) against the current dataset that was analyzed. The Amazon SageMaker Model Monitor prebuilt container provides the following violation checks.

```
{
  "violations": [
    {
      "feature_name" : "string",
      "constraint_check_type" :
        "data_type_check",
        | "completeness_check",
        | "baseline_drift_check",
        | "missing_column_check",
        | "extra_column_check",
        | "categorical_values_check"
      "description" : "string"
    }
  ]
}
```

### Types of Violations Monitored

Violation Check Type	Description
<code>data_type_check</code>	If the data types in the current execution are not the same as in the baseline dataset, this violation is flagged.  During the baseline step, the generated constraints suggest the inferred data type for each column. The <code>monitoring_config.datatype_check_threshold</code> parameter can be tuned to adjust the threshold on when it is flagged as a violation.
<code>completeness_check</code>	If the completeness (% of non-null items) observed in the current execution exceeds the

Violation Check Type	Description
	threshold specified in completeness threshold specified per feature, this violation is flagged.
	During the baseline step, the generated constraints suggest a completeness value.
<code>baseline_drift_check</code>	If the calculated distribution distance between the current and the baseline datasets is more than the threshold specified in <code>monitoring_config.comparison_threshold</code> , this violation is flagged.
<code>missing_column_check</code>	If the number of columns in the current dataset is less than the number in the baseline dataset, this violation is flagged.
<code>extra_column_check</code>	If the number of columns in the current dataset is more than the number in the baseline, this violation is flagged.
<code>categorical_values_check</code>	If there are more unknown values in the current dataset than in the baseline dataset, this violation is flagged. This value is dictated by the threshold in <code>monitoring_config.domain_content_threshold</code> .

## Monitor Model Quality

Model quality monitoring jobs monitor the performance of a model by comparing the predictions that the model makes with the actual ground truth labels that the model attempts to predict. To do this, model quality monitoring merges data that is captured from real-time inference with actual labels that you store in an Amazon S3 bucket, and then compares the predictions with the actual labels.

To measure model quality, model monitor uses metrics that depend on the ML problem type. For example, if your model is for a regression problem, one of the metrics evaluated is mean square error (mse). For information about all of the metrics used for the different ML problem types, see [Model Quality Metrics \(p. 1966\)](#).

Model quality monitoring follows the same steps as data quality monitoring, but adds the additional step of merging the actual labels from Amazon S3 with the predictions captured from the real-time inference endpoint. To monitor model quality, follow these steps:

- Enable data capture. This captures inference input and output from a real-time inference endpoint and stores the data in Amazon S3. For more information, see [Capture Data \(p. 1977\)](#).
- Create a baseline. In this step, you run a baseline job that compares predictions from the model with ground truth labels in a baseline dataset. The baseline job automatically creates baseline statistical rules and constraints that define thresholds against which the model performance is evaluated. For more information, see [Create a Model Quality Baseline \(p. 1964\)](#).
- Define and schedule model quality monitoring jobs. For more information, see [Schedule Model Quality Monitoring Jobs \(p. 1965\)](#).
- Ingest ground truth labels that model monitor merges with captured prediction data from real-time inference endpoint. For more information, see [Ingest Ground Truth Labels and Merge Them With Predictions \(p. 1966\)](#).

- Integrate model quality monitoring with Amazon CloudWatch. For more information, see [Model Quality CloudWatch Metrics \(p. 1970\)](#).
- Interpret the results of a monitoring job. For more information, see [Interpret Results \(p. 1982\)](#).
- Use SageMaker Studio to enable model quality monitoring and visualize results. For more information, see [Visualize Results in Amazon SageMaker Studio \(p. 1984\)](#).

## Topics

- [Create a Model Quality Baseline \(p. 1964\)](#)
- [Schedule Model Quality Monitoring Jobs \(p. 1965\)](#)
- [Ingest Ground Truth Labels and Merge Them With Predictions \(p. 1966\)](#)
- [Model Quality Metrics \(p. 1966\)](#)
- [Model Quality CloudWatch Metrics \(p. 1970\)](#)

## Create a Model Quality Baseline

Create a baseline job that compares your model predictions with ground truth labels in a baseline dataset that you have stored in Amazon S3. Typically, you use a training dataset as the baseline dataset. The baseline job calculates metrics for the model and suggests constraints to use to monitor model quality drift.

To create a baseline job, you need to have a dataset that contains predictions from your model along with labels that represent the ground truth for your data.

To create a baseline job use the `ModelQualityMonitor` class provided by the SageMaker Python SDK, and complete the following steps.

### To create a model quality baseline job

1. First, create an instance of the `ModelQualityMonitor` class. The following code snippet shows how to do this.

```
from sagemaker import get_execution_role, session, Session
from sagemaker.model_monitor import ModelQualityMonitor

role = get_execution_role()
session = Session()

model_quality_monitor = ModelQualityMonitor(
    role=role,
    instance_count=1,
    instance_type='ml.m5.xlarge',
    volume_size_in_gb=20,
    max_runtime_in_seconds=1800,
    sagemaker_session=session
)
```

2. Now call the `suggest_baseline` method of the `ModelQualityMonitor` object to run a baseline job. The following code snippet assumes that you have a baseline dataset that contains both predictions and labels stored in Amazon S3.

```
baseline_job_name = "MyBaselineJob"
job = model_quality_monitor.suggest_baseline(
    job_name=baseline_job_name,
    baseline_dataset=baseline_dataset_uri, # The S3 location of the validation dataset.
    dataset_format=DatasetFormat.csv(header=True),
```

```
        output_s3_uri = baseline_results_uri, # The S3 location to store the results.
        problem_type='BinaryClassification',
        inference_attribute= "prediction", # The column in the dataset that contains
predictions.
        probability_attribute= "probability", # The column in the dataset that contains
probabilities.
        ground_truth_attribute= "label" # The column in the dataset that contains ground
truth labels.
)
job.wait(logs=False)
```

3. After the baseline job finishes, you can see the constraints that the job generated. First, get the results of the baseline job by calling the `latest_baselining_job` method of the `ModelQualityMonitor` object.

```
baseline_job = model_quality_monitor.latest_baselining_job
```

4. The baseline job suggests constraints, which are thresholds for metrics that model monitor measures. If a metric goes beyond the suggested threshold, Model Monitor reports a violation. To view the constraints that the baseline job generated, call the `suggested_constraints` method of the baseline job. The following code snippet loads the constraints for a binary classification model into a Pandas dataframe.

```
import pandas as pd
pd.DataFrame(baseline_job.suggested_constraints().body_dict["binary_classification_constraints"]).T
```

We recommend that you view the generated constraints and modify them as necessary before using them for monitoring. For example, if a constraint is too aggressive, you might get more alerts for violations than you want.

5. When you are satisfied with the constraints, pass them as the `constraints` parameter when you create a monitoring schedule. For more information, see [Schedule Model Quality Monitoring Jobs \(p. 1965\)](#).

The suggested baseline constraints are contained in the `constraints.json` file in the location you specify with `output_s3_uri`. For information about the schema for this file in the [Schema for Constraints \(constraints.json file\) \(p. 1997\)](#).

## Schedule Model Quality Monitoring Jobs

For general information about scheduling monitoring jobs, see [the section called “Schedule Monitoring Jobs” \(p. 1979\)](#). For model quality monitoring, you also have to consider the fact that the availability of ground truth labels might be delayed.

To address this, use offsets. Model quality jobs include `StartOffset` and `EndOffset`, which are fields of the `ModelQualityJobInput` parameter of the `create_model_quality_job_definition` method that work as follows:

- `StartOffset` - If specified, jobs subtract this time from the start time.
- `EndOffset` - If specified, jobs subtract this time from the end time.

The format of the offsets are, for example, `-P7H`, where `7H` is 7 hours. You can use `-P#H` or `-P#D`, where `H=hours`, `D=days`, and `M=minutes`, and `#` is the number.

For example, if your ground truth starts coming in after 1 day, but is not complete for a week, set `StartOffset` to `-P8D` and `EndOffset` to `-P1D`. Then, if you schedule a job to run at `2020-01-09T13:00`, it analyzes data from between `2020-01-01T13:00` and `2020-01-08T13:00`.

### Important

The schedule cadence should be such that one execution finishes before the next execution starts, which allows the ground truth merge job and monitoring job from the execution to complete. The maximum runtime of an execution is divided between the two jobs, so for an hourly model quality monitoring job, the value of `MaxRuntimeInSeconds` specified as part of `StoppingCondition` should be no more than 1800.

## Ingest Ground Truth Labels and Merge Them With Predictions

Model quality monitoring compares the predictions your model makes with ground truth labels to measure the quality of the model. For this to work, you periodically label data captured by your endpoint and upload it to Amazon S3.

To match ground truth labels with captured prediction data, there must be a unique identifier for each record in the dataset. The structure of each record for ground truth data is as follows:

```
{  
    "groundTruthData": {  
        "data": "1",  
        "encoding": "CSV" # only CSV supported at launch, we assume "data" only consists of  
        "label"  
    },  
    "eventMetadata": {  
        "eventId": "aaaa-bbbb-cccc"  
    },  
    "eventVersion": "0"  
}
```

In the `groundTruthData` structure, `eventId` can be one of the following:

- `eventId` – This ID is automatically generated when a user invokes the endpoint.
- `inferenceId` – The caller supplies this ID when they invoke the endpoint.

If `inferenceId` is present in captured data records, Model Monitor uses it to merge captured data with ground truth records. You are responsible for making sure that the `inferenceId` in the ground truth records match the `inferenceId` in the captured records. If `inferenceId` is not present in captured data, model monitor uses `eventId` from the captured data records to match them with a ground truth record.

You must upload ground truth data to an Amazon S3 bucket that has the same path format as captured data, which is of the following form:

```
s3://bucket/prefixyyyy/mm/dd/hh
```

The date in this path is the date when the ground truth label is collected, and does not have to match the date when the inference was generated.

After you create and upload the ground truth labels, include the location of the labels as a parameter when you create the monitoring job. If you are using Amazon SDK for Python (Boto3), do this by specifying the location of ground truth labels as the `S3Uri` field of the `GroundTruthS3Input` parameter in a call to the `create_model_quality_job_definition` method. If you are using the SageMaker Python SDK, specify the location of the ground truth labels as the `ground_truth_input` parameter in the call to the `create_monitoring_schedule` of the `ModelQualityMonitor` object.

## Model Quality Metrics

Model quality monitoring jobs compute different metrics depending on the ML problem type. The following sections list the metrics analyzed for each ML problem type.

### Note

Standard deviation for metrics are provided only when at least 200 samples are available. Model Monitor computes standard deviation by randomly sampling 80% of the data 5 times, computing the metric, and taking the standard deviation for those results.

## Regression Metrics

The following shows an example of the metrics that model quality monitor computes for a regression problem.

```
"regression_metrics" : {
    "mae" : {
        "value" : 0.3711832061068702,
        "standard_deviation" : 0.0037566388129940394
    },
    "mse" : {
        "value" : 0.3711832061068702,
        "standard_deviation" : 0.0037566388129940524
    },
    "rmse" : {
        "value" : 0.609248066149471,
        "standard_deviation" : 0.003079253267651125
    },
    "r2" : {
        "value" : -1.3766111872212665,
        "standard_deviation" : 0.022653980022771227
    }
}
```

## Binary Classification Metrics

The following shows an example of the metrics that model quality monitor computes for a binary classification problem.

```
"binary_classification_metrics" : {
    "confusion_matrix" : {
        "0" : {
            "0" : 1,
            "1" : 2
        },
        "1" : {
            "0" : 0,
            "1" : 1
        }
    },
    "recall" : {
        "value" : 1.0,
        "standard_deviation" : "NaN"
    },
    "precision" : {
        "value" : 0.3333333333333333,
        "standard_deviation" : "NaN"
    },
    "accuracy" : {
        "value" : 0.5,
        "standard_deviation" : "NaN"
    },
    "recall_best_constant_classifier" : {
        "value" : 1.0,
        "standard_deviation" : "NaN"
    },
    "precision_best_constant_classifier" : {
        "value" : 0.25,
        "standard_deviation" : "NaN"
    }
}
```

```

        "standard_deviation" : "NaN"
    },
    "accuracy_best_constant_classifier" : {
        "value" : 0.25,
        "standard_deviation" : "NaN"
    },
    "true_positive_rate" : {
        "value" : 1.0,
        "standard_deviation" : "NaN"
    },
    "true_negative_rate" : {
        "value" : 0.3333333333333337,
        "standard_deviation" : "NaN"
    },
    "false_positive_rate" : {
        "value" : 0.6666666666666666,
        "standard_deviation" : "NaN"
    },
    "false_negative_rate" : {
        "value" : 0.0,
        "standard_deviation" : "NaN"
    },
    "receiver_operating_characteristic_curve" : {
        "false_positive_rates" : [ 0.0, 0.0, 0.0, 0.0, 0.0, 1.0 ],
        "true_positive_rates" : [ 0.0, 0.25, 0.5, 0.75, 1.0, 1.0 ]
    },
    "precision_recall_curve" : {
        "precisions" : [ 1.0, 1.0, 1.0, 1.0, 1.0 ],
        "recalls" : [ 0.0, 0.25, 0.5, 0.75, 1.0 ]
    },
    "auc" : {
        "value" : 1.0,
        "standard_deviation" : "NaN"
    },
    "f0_5" : {
        "value" : 0.3846153846153846,
        "standard_deviation" : "NaN"
    },
    "f1" : {
        "value" : 0.5,
        "standard_deviation" : "NaN"
    },
    "f2" : {
        "value" : 0.7142857142857143,
        "standard_deviation" : "NaN"
    },
    "f0_5_best_constant_classifier" : {
        "value" : 0.29411764705882354,
        "standard_deviation" : "NaN"
    },
    "f1_best_constant_classifier" : {
        "value" : 0.4,
        "standard_deviation" : "NaN"
    },
    "f2_best_constant_classifier" : {
        "value" : 0.625,
        "standard_deviation" : "NaN"
    }
}
}

```

## Multiclass Metrics

The following shows an example of the metrics that model quality monitor computes for a multiclass classification problem.

```
"multiclass_classification_metrics" : {
    "confusion_matrix" : {
        "0" : {
            "0" : 1180,
            "1" : 510
        },
        "1" : {
            "0" : 268,
            "1" : 138
        }
    },
    "accuracy" : {
        "value" : 0.6288167938931297,
        "standard_deviation" : 0.00375663881299405
    },
    "weighted_recall" : {
        "value" : 0.6288167938931297,
        "standard_deviation" : 0.003756638812994008
    },
    "weighted_precision" : {
        "value" : 0.6983172269629505,
        "standard_deviation" : 0.006195912915307507
    },
    "weighted_f0_5" : {
        "value" : 0.6803947317178771,
        "standard_deviation" : 0.005328406973561699
    },
    "weighted_f1" : {
        "value" : 0.6571162346664904,
        "standard_deviation" : 0.004385008075019733
    },
    "weighted_f2" : {
        "value" : 0.6384024354394601,
        "standard_deviation" : 0.003867109755267757
    },
    "accuracy_best_constant_classifier" : {
        "value" : 0.19370229007633588,
        "standard_deviation" : 0.0032049848450732355
    },
    "weighted_recall_best_constant_classifier" : {
        "value" : 0.19370229007633588,
        "standard_deviation" : 0.0032049848450732355
    },
    "weighted_precision_best_constant_classifier" : {
        "value" : 0.03752057718081697,
        "standard_deviation" : 0.001241536088657851
    },
    "weighted_f0_5_best_constant_classifier" : {
        "value" : 0.04473443104152011,
        "standard_deviation" : 0.0014460485504284792
    },
    "weighted_f1_best_constant_classifier" : {
        "value" : 0.06286421244683643,
        "standard_deviation" : 0.0019113576884608862
    },
    "weighted_f2_best_constant_classifier" : {
        "value" : 0.10570313141262414,
        "standard_deviation" : 0.002734216826748117
    }
}
```

## Model Quality CloudWatch Metrics

If you set the value of the `enable_cloudwatch_metrics` to `True` when you create the monitoring schedule, model quality monitoring jobs send all metrics to Amazon CloudWatch.

Model quality metrics appear in the `aws/sagemaker/Endpoints/model-metrics` namespace. For a list of the metrics that are emitted, see [Model Quality Metrics \(p. 1966\)](#).

You can use CloudWatch metrics to create an alarm when a specific metric doesn't meet the threshold you specify. For instructions about how to create CloudWatch alarms, see [Create a CloudWatch Alarm Based on a Static Threshold](#) in the *Amazon CloudWatch User Guide*.

## Monitor Bias Drift for Models in Production

Amazon SageMaker Clarify bias monitoring helps data scientists and ML engineers monitor predictions for bias on a regular basis. As the model is monitored, customers can view exportable reports and graphs detailing bias in SageMaker Studio and configure alerts in Amazon CloudWatch to receive notifications if bias beyond a certain threshold is detected. Bias can be introduced or exacerbated in deployed ML models when the training data differs from the data that the model sees during deployment (that is, the live data). These kinds of changes in the live data distribution might be temporary (for example, due to some short-lived, real-world events) or permanent. In either case, it might be important to detect these changes. For example, the outputs of a model for predicting home prices can become biased if the mortgage rates used to train the model differ from current, real-world mortgage rates. With bias detection capabilities in Model Monitor, when SageMaker detects bias beyond a certain threshold, it automatically generates metrics that you can view in SageMaker Studio and through Amazon CloudWatch alerts.

In general, measuring bias only during the train-and-deploy phase might not be sufficient. It is possible that after the model has been deployed, the distribution of the data that the deployed model sees (that is, the live data) is different from data distribution in the training dataset. This change might introduce bias in a model over time. The change in the live data distribution might be temporary (for example, due to some short-lived behavior like the holiday season) or permanent. In either case, it might be important to detect these changes and take steps to reduce the bias when appropriate.

To detect these changes, SageMaker Clarify provides functionality to monitor the bias metrics of a deployed model continuously and raise automated alerts if the metrics exceed a threshold. For example, consider the DPPL bias metric. Specify an allowed range of values  $A=(a_{\min}, a_{\max})$ , for instance an interval of  $(-0.1, 0.1)$ , that DPPL should belong to during deployment. Any deviation from this range should raise a *bias detected* alert. With SageMaker Clarify, you can perform these checks at regular intervals.

For example, you can set the frequency of the checks to 2 days. This means that SageMaker Clarify computes the DPPL metric on data collected during a 2-day window. In this example,  $D_{\text{win}}$  is the data that the model processed during last 2-day window. An alert is issued if the DPPL value  $b_{\text{win}}$  computed on  $D_{\text{win}}$  falls outside of an allowed range  $A$ . This approach to checking if  $b_{\text{win}}$  is outside of  $A$  can be somewhat noisy.  $D_{\text{win}}$  might consist of very few samples and might not be representative of the live data distribution. The small sample size means that the value of bias  $b_{\text{win}}$  computed over  $D_{\text{win}}$  might not be a very robust estimate. In fact, very high (or low) values of  $b_{\text{win}}$  may be observed purely due to chance. To ensure that the conclusions drawn from the observed data  $D_{\text{win}}$  are statistically significant, SageMaker Clarify makes use of confidence intervals. Specifically, it uses the Normal Bootstrap Interval method to construct an interval  $C=(c_{\min}, c_{\max})$  such that SageMaker Clarify is confident that the true bias value computed over the full live data is contained in  $C$  with high probability. Now, if the confidence interval  $C$  overlaps with the allowed range  $A$ , SageMaker Clarify interprets it as "it is likely that the bias metric value of the live data distribution falls within the allowed range". If  $C$  and  $A$  are disjoint, SageMaker Clarify is confident that the bias metric does not lie in  $A$  and raises an alert.

## Model Monitor Sample Notebook

Amazon SageMaker Clarify provides the following sample notebook that shows how to capture real-time inference data, create a baseline to monitor evolving bias against, and inspect the results:

- [Monitoring bias drift and feature attribution drift Amazon SageMaker Clarify](#) – Use Amazon SageMaker Model Monitor to monitor bias drift and feature attribution drift over time.

This notebook has been verified to run in Amazon SageMaker Studio only. If you need instructions on how to open a notebook in Amazon SageMaker Studio, see [Create or Open an Amazon SageMaker Studio Notebook \(p. 79\)](#). If you're prompted to choose a kernel, choose **Python 3 (Data Science)**. The following topics contain the highlights from the last two steps, and they contain code examples from the example notebook.

### Topics

- [Create a Bias Drift Baseline \(p. 1971\)](#)
- [Schedule Bias Drift Monitoring Jobs \(p. 1972\)](#)
- [Inspect Reports for Data Bias Drift \(p. 1973\)](#)

## Create a Bias Drift Baseline

After you have configured your application to capture real-time inference data, the first task to monitor for bias drift is to create a baseline. This involves configuring the data inputs, which groups are sensitive, how the predictions are captured, and the model and its posttraining bias metrics. Then you need to start the baselining job.

Model bias monitor can detect bias drift of ML models on a regular basis. Similar to the other monitoring types, the standard procedure of creating a model bias monitor is first baselining and then establishing a monitoring schedule.

```
model_bias_monitor = ModelBiasMonitor(  
    role=role,  
    sagemaker_session=sagemaker_session,  
    max_runtime_in_seconds=1800,  
)
```

DataConfig stores information about the dataset to be analyzed (for example, the dataset file), its format (that is, CSV or JSONLines), headers (if any) and label.

```
model_bias_baselining_job_result_uri = f"{baseline_results_uri}/model_bias"  
model_bias_data_config = DataConfig(  
    s3_data_input_path=validation_dataset,  
    s3_output_path=model_bias_baselining_job_result_uri,  
    label=label_header,  
    headers=all_headers,  
    dataset_type=dataset_type,  
)
```

BiasConfig is the configuration of the sensitive groups in the dataset. Typically, bias is measured by computing a metric and comparing it across groups. The group of interest is called the *facet*. For posttraining bias, you should also take the positive label into account.

```
model_bias_config = BiasConfig(
```

```

        label_values_or_threshold=[1],
        facet_name="Account Length",
        facet_values_or_threshold=[100],
    )
)

```

`ModelPredictedLabelConfig` specifies how to extract a predicted label from the model output. In this example, the 0.8 cutoff has been chosen in anticipation that customers will turn over frequently. For more complicated outputs, there are a few more options, like "label" is the index, name, or JSONPath to locate predicted label in endpoint response payload.

```

model_predicted_label_config = ModelPredictedLabelConfig(
    probability_threshold=0.8,
)

```

`ModelConfig` is the configuration related to the model to be used for inferencing. In order to compute posttraining bias metrics, the computation needs to get inferences for the model name provided. To accomplish this, the processing job uses the model to create an ephemeral endpoint (also known as *shadow endpoint*). The processing job deletes the shadow endpoint after the computations are completed. This configuration is also used by the explainability monitor.

```

model_config = ModelConfig(
    model_name=model_name,
    instance_count=endpoint_instance_count,
    instance_type=endpoint_instance_type,
    content_type=dataset_type,
    accept_type=dataset_type,
)

```

Now you can start the baselining job.

```

model_bias_monitor.suggest_baseline(
    model_config=model_config,
    data_config=model_bias_data_config,
    bias_config=model_bias_config,
    model_predicted_label_config=model_predicted_label_config,
)
print(f"ModelBiasMonitor baselining job: {model_bias_monitor.latest_baselining_job_name}")

```

The scheduled monitor automatically picks up baselining job name and waits for it before monitoring begins.

## Schedule Bias Drift Monitoring Jobs

Now that you have a baseline, you can call the `create_monitoring_schedule()` method to schedule an hourly monitor to analyze the data with a monitoring schedule. If you have submitted a baselining job, the monitor automatically picks up analysis configuration from the baselining job. If you skip the baselining step or the capture dataset has a different nature from the training dataset, you must provide the analysis configuration.

```

model_bias_analysis_config = None
if not model_bias_monitor.latest_baselining_job:
    model_bias_analysis_config = BiasAnalysisConfig(
        model_bias_config,
        headers=all_headers,
        label=label_header,
    )
model_bias_monitor.create_monitoring_schedule(
    analysis_config=model_bias_analysis_config,
    output_s3_uri=s3_report_path,
)

```

```

endpoint_input=EndpointInput(
    endpoint_name=endpoint_name,
    destination="/opt/ml/processing/input/endpoint",
    start_time_offset="-PT1H",
    end_time_offset="-PT0H",
    probability_threshold_attribute=0.8,
),
ground_truth_input=ground_truth_upload_path,
schedule_cron_expression=schedule_expression,
)
print(f"Model bias monitoring schedule: {model_bias_monitor.monitoring_schedule_name}")

```

## Inspect Reports for Data Bias Drift

If you are not able to inspect the results of the monitoring in the generated reports in SageMaker Studio, you can print them out as follows:

```

schedule_desc = model_bias_monitor.describe_schedule()
execution_summary = schedule_desc.get("LastMonitoringExecutionSummary")
if execution_summary and execution_summary["MonitoringExecutionStatus"] in ["Completed",
    "CompletedWithViolations"]:
    last_model_bias_monitor_execution = model_bias_monitor.list_executions()[-1]
    last_model_bias_monitor_execution_report_uri =
    last_model_bias_monitor_execution.output.destination
    print(f'Report URI: {last_model_bias_monitor_execution_report_uri}')
    last_model_bias_monitor_execution_report_files =
    sorted(S3Downloader.list(last_model_bias_monitor_execution_report_uri))
    print("Found Report Files:")
    print("\n ".join(last_model_bias_monitor_execution_report_files))
else:
    last_model_bias_monitor_execution = None
    print("====STOP==== \n No completed executions to inspect further. Please wait till an
execution completes or investigate previously reported failures.")

```

If there are violations compared to the baseline, they are listed here:

```

if last_model_bias_monitor_execution:
    model_bias_violations = last_model_bias_monitor_execution.constraintViolations()
    if model_bias_violations:
        print(model_bias_violations.body_dict)

```

In SageMaker Studio, you can see visualizations of the analysis results and CloudWatch metrics by choosing the **Endpoints** tab, and then double-clicking the endpoint.

## Monitor Feature Attribution Drift for Models in Production

A drift in the distribution of live data for models in production can result in a corresponding drift in the feature attribution values, just as it could cause a drift in bias when monitoring bias metrics. Amazon SageMaker Clarify feature attribution monitoring helps data scientists and ML engineers monitor predictions for feature attribution drift on a regular basis. As the model is monitored, customers can view exportable reports and graphs detailing feature attributions in SageMaker Studio and configure alerts in Amazon CloudWatch to receive notifications if it is detected that the attribution values drift beyond a certain threshold.

To illustrate this with a specific situation, consider a hypothetical scenario for college admissions. Assume that we observe the following (aggregated) feature attribution values in the training data and in the live data:

### College Admission Hypothetical Scenario

Feature	Attribution in training data	Attribution in live data
SAT score	0.70	0.10
GPA	0.50	0.20
Class rank	0.05	0.70

The change from training data to live data appears significant. The feature ranking has completely reversed. Similar to the bias drift, the feature attribution drifts might be caused by a change in the live data distribution and warrant a closer look into the model behavior on the live data. Again, the first step in these scenarios is to raise an alarm that a drift has happened.

We can detect the drift by comparing how the ranking of the individual features changed from training data to live data. In addition to being sensitive to changes in ranking order, we also want to be sensitive to the raw attribution score of the features. For instance, given two features that fall in the ranking by the same number of positions going from training to live data, we want to be more sensitive to the feature that had a higher attribution score in the training data. With these properties in mind, we use the Normalized Discounted Cumulative Gain (NDCG) score for comparing the feature attributions rankings of training and live data.

Specifically, assume we have the following:

- $F=[f_1, \dots, f_m]$  is the list of features sorted with respect to their attribution scores in the training data where  $m$  is the total number of features. For instance, in our case,  $F=[\text{SAT Score}, \text{GPA}, \text{Class Rank}]$ .
- $a(f)$  is a function that returns the feature attribution score on the training data given a feature  $f$ . For example,  $a(\text{SAT Score}) = 0.70$ .
- $F'=[f'_1, \dots, f'_m]$  is the list of features sorted with respect to their attribution scores in the live data. For example,  $F'=[\text{Class Rank}, \text{GPA}, \text{SAT Score}]$ .

Then, we can compute the NDCG as:

$$\text{NDCG}=\text{DCG}/\text{iDCG}$$

with

- $\text{DCG} = \sum_1^m a(f'_i)/\log_2(i+1)$
- $\text{iDCG} = \sum_1^m a(f_i)/\log_2(i+1)$

The quantity DCG measures whether features with high attribution in the training data are also ranked higher in the feature attribution computed on the live data. The quantity iDCG measures the *ideal score* and it's just a normalizing factor to ensure that the final quantity resides in the range  $[0, 1]$ , with 1 being the best possible value. A NDCG value of 1 means that the feature attribution ranking in the live data is the same as the one in the training data. In this particular example, because the ranking changed by quite a bit, the NDCG value is 0.69.

In SageMaker Clarify, if the NDCG value is below 0.90, we automatically raise an alert.

## Model Monitor Example Notebook

SageMaker Clarify provides the following example notebook that shows how to capture real-time inference data, create a baseline to monitor evolving bias against, and inspect the results:

- [Monitoring bias drift and feature attribution drift Amazon SageMaker Clarify](#) – Use Amazon SageMaker Model Monitor to monitor bias drift and feature attribution drift over time.

This notebook has been verified to run in SageMaker Studio only. If you need instructions on how to open a notebook in SageMaker Studio, see [Create or Open an Amazon SageMaker Studio Notebook \(p. 79\)](#). If you're prompted to choose a kernel, choose **Python 3 (Data Science)**. The following topics contain the highlights from the last two steps, and they contain code examples from the example notebook.

### Topics

- [Create a SHAP Baseline for Models in Production \(p. 1975\)](#)
- [Schedule Feature Attribute Drift Monitoring Jobs \(p. 1976\)](#)
- [Inspect Reports for Feature Attribute Drift in Production Models \(p. 1976\)](#)

## Create a SHAP Baseline for Models in Production

Explanations are typically contrastive, that is, they account for deviations from a baseline. For information on explainability baselines, see [SHAP Baselines for Explainability \(p. 1854\)](#).

In addition to providing explanations for per-instance inferences, SageMaker Clarify also supports global explanation for ML models that helps you understand the behavior of a model as a whole in terms of its features. SageMaker Clarify generates a global explanation of an ML model by aggregating the Shapley values over multiple instances. SageMaker Clarify supports the following different ways of aggregation, which you can use to define baselines:

- `mean_abs` – Mean of absolute SHAP values for all instances.
- `median` – Median of SHAP values for all instances.
- `mean_sq` – Mean of squared SHAP values for all instances.

After you have configured your application to capture real-time inference data, the first task to monitor for drift in feature attribution is to create a baseline to compare against. This involves configuring the data inputs, which groups are sensitive, how the predictions are captured, and the model and its posttraining bias metrics. Then you need to start the baselining job. Model explainability monitor can explain the predictions of a deployed model that's producing inferences and detect feature attribution drift on a regular basis.

```
model_explainability_monitor = ModelExplainabilityMonitor(  
    role=role,  
    sagemaker_session=sagemaker_session,  
    max_runtime_in_seconds=1800,  
)
```

In this example, the explainability baselining job shares the test dataset with the bias baselining job, so it uses the same `DataConfig`, and the only difference is the job output URI.

```
model_explainability_baselining_job_result_uri = f"{baseline_results_uri}/  
model_explainability"  
model_explainability_data_config = DataConfig(  
    s3_data_input_path=validation_dataset,  
    s3_output_path=model_explainability_baselining_job_result_uri,  
    label=label_header,  
    headers=all_headers,  
    dataset_type=dataset_type,  
)
```

Currently the SageMaker Clarify explainer offers a scalable and efficient implementation of SHAP, so the explainability config is `SHAPConfig`, including the following:

- **baseline** – A list of rows (at least one) or S3 object URI to be used as the baseline dataset in the Kernel SHAP algorithm. The format should be the same as the dataset format. Each row should contain only the feature columns/values and omit the label column/values.
- **num\_samples** – Number of samples to be used in the Kernel SHAP algorithm. This number determines the size of the generated synthetic dataset to compute the SHAP values.
- **agg\_method** – Aggregation method for global SHAP values. Following are valid values:
  - **mean\_abs** – Mean of absolute SHAP values for all instances.
  - **median** – Median of SHAP values for all instances.
  - **mean\_sq** – Mean of squared SHAP values for all instances.
- **use\_logit** – Indicator of whether the logit function is to be applied to the model predictions. Default is `False`. If `use_logit` is `True`, the SHAP values will have log-odds units.
- **save\_local\_shap\_values (bool)** – Indicator of whether to save the local SHAP values in the output location. Default is `True`.

```
# Here use the mean value of test dataset as SHAP baseline
test_dataframe = pd.read_csv(test_dataset, header=None)
shap_baseline = [list(test_dataframe.mean())]

shap_config = SHAPConfig(
    baseline=shap_baseline,
    num_samples=100,
    agg_method="mean_abs",
    save_local_shap_values=False,
)
```

Start a baselining job. The same `model_config` is required because the explainability baselining job needs to create a shadow endpoint to get predictions for the generated synthetic dataset.

```
model_explainability_monitor.suggest_baseline(
    data_config=model_explainability_data_config,
    model_config=model_config,
    explainability_config=shap_config,
)
print(f"ModelExplainabilityMonitor baselining job:
{model_explainability_monitor.latest_baselining_job_name}")
```

## Schedule Feature Attribute Drift Monitoring Jobs

Model explainability monitoring helps you understand and interpret the predictions made by your ML models. When Model Monitor is configured to monitor model explainability, SageMaker automatically detects any drift in relative importance of features and creates reports explaining feature attributions.

Call the `create_monitoring_schedule()` method to schedule an hourly monitor to analyze the data with a monitoring schedule. If a baselining job has been submitted, the monitor automatically picks up analysis configuration from the baselining job. However, if you skip the baselining step or the capture dataset has a different nature from the training dataset, you have to provide the analysis configuration. `ModelConfig` is required by `ExplainabilityAnalysisConfig` for the same reason that it's required for the baselining job. Note that only features are required for computing feature attribution, so you should exclude ground truth labeling.

## Inspect Reports for Feature Attribute Drift in Production Models

After the schedule that you set up is started by default, you need to wait for its first execution to start, and then stop the schedule to avoid incurring charges.

To inspect the reports, use the following code:

```

schedule_desc = model_explainability_monitor.describe_schedule()
execution_summary = schedule_desc.get("LastMonitoringExecutionSummary")
if execution_summary and execution_summary["MonitoringExecutionStatus"] in ["Completed",
    "CompletedWithViolations"]:
    last_model_explainability_monitor_execution =
        model_explainability_monitor.list_executions()[-1]
    last_model_explainability_monitor_execution_report_uri =
        last_model_explainability_monitor_execution.output.destination
    print(f'Report URI: {last_model_explainability_monitor_execution_report_uri}')
    last_model_explainability_monitor_execution_report_files =
        sorted(S3Downloader.list(last_model_explainability_monitor_execution_report_uri))
    print("Found Report Files:")
    print("\n ".join(last_model_explainability_monitor_execution_report_files))
else:
    last_model_explainability_monitor_execution = None
    print("====STOP==== \n No completed executions to inspect further. Please wait till an
execution completes or investigate previously reported failures.")

```

If there are any violations compared to the baseline, they are listed here:

```

if last_model_explainability_monitor_execution:
    model_explainability_violations =
        last_model_explainability_monitor_execution.constraintViolations()
    if model_explainability_violations:
        print(model_explainability_violations.body_dict)

```

In SageMaker Studio, you can see visualizations of the analysis results and CloudWatch metrics by choosing the **Endpoints** tab, and then double-clicking the endpoint.

## Capture Data

To use Model Monitor, configure your real-time inference endpoint to capture data from requests and responses and store the captured data in Amazon S3. Model monitor compares metrics from this data with a baseline that you create for the model. For information about creating a baseline, see [Create a Baseline \(p. 1959\)](#).

### Note

To prevent impact to inference requests, Data Capture stops capturing requests at high levels of disk usage. It is recommended you keep your disk utilization below 75% in order to ensure data capture continues capturing requests.

### To set up data capture

1. First, configure the Amazon S3 buckets Model Monitor uses to store the captured data.

```

import boto3
import re
import json
from sagemaker import get_execution_role, session

region= boto3.Session().region_name

role = get_execution_role()
print("RoleArn: {}".format(role))

# You can use a different bucket, but make sure the role you chose for this notebook
# has s3:PutObject permissions. This is the bucket into which the data is captured
bucket = session.Session(boto3.Session()).default_bucket()
print("Demo Bucket: {}".format(bucket))
prefix = 'sagemaker/DEMO-ModelMonitor'

```

```

data_capture_prefix = '{}/datacapture'.format(prefix)
s3_capture_upload_path = 's3://{}{}'.format(bucket, data_capture_prefix)
reports_prefix = '{}/reports'.format(prefix)
s3_report_path = 's3://{}{}'.format(bucket, reports_prefix)
code_prefix = '{}/code'.format(prefix)
s3_code_preprocessor_uri = 's3://{}{}'.format(bucket, code_prefix,
    'preprocessor.py')
s3_code_postprocessor_uri = 's3://{}{}'.format(bucket, code_prefix,
    'postprocessor.py')

print("Capture path: {}".format(s3_capture_upload_path))
print("Report path: {}".format(s3_report_path))
print("Preproc Code path: {}".format(s3_code_preprocessor_uri))
print("Postproc Code path: {}".format(s3_code_postprocessor_uri))

```

2. Next, upload the pre-trained model to Amazon S3.

```

model_file = open("model/your-prediction-model.tar.gz", 'rb')
s3_key = os.path.join(prefix, 'your-prediction-model.tar.gz')
boto3.Session().resource('s3').Bucket(bucket).Object(s3_key).upload_fileobj(model_file)

```

3. Configure the data you want to capture by configuring the data you want to capture in a [DataCaptureConfig](#) structure. You can capture the request payload, the response payload, or both with this configuration.

```

from sagemaker.model_monitor import DataCaptureConfig

endpoint_name = 'your-pred-model-monitor-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("EndpointName={}".format(endpoint_name))

data_capture_config=DataCaptureConfig(
    enable_capture = True,
    sampling_percentage=100,
    destination_s3_uri=s3_capture_upload_path)

```

4. Enable data capture by passing the configuration you created in the previous step when you deploy the model.

```

predictor = model.deploy(initial_instance_count=1,
    instance_type='ml.m4.xlarge',
    endpoint_name=endpoint_name
    data_capture_config=data_capture_config)

```

5. Invoke the endpoint to send data to the endpoint to get inferences in real time. Because you enabled the data capture in the previous steps, the request and response payload, along with some additional metadata, is saved in the Amazon S3 location that you specified in [DataCaptureConfig](#).

```

from sagemaker.predictor import RealTimePredictor
import time

predictor = RealTimePredictor(endpoint=endpoint_name,content_type='text/csv')

# get a subset of test data for a quick test
!head -120 test_data/test-dataset-input-cols.csv > test_data/test_sample.csv
print("Sending test traffic to the endpoint {}. \nPlease
wait...".format(endpoint_name))

with open('test_data/test_sample.csv', 'r') as f:
    for row in f:
        payload = row.rstrip('\n')
        response = predictor.predict(data=payload)

```

```

    time.sleep(0.5)
    print("Done!")

```

6. View captured data by listing the data capture files stored in Amazon S3. Expect to see different files from different time periods, organized based on the hour when the invocation occurred.

```

s3_client = boto3.Session().client('s3')
current_endpoint_capture_prefix = '{} / {}'.format(data_capture_prefix, endpoint_name)
result = s3_client.list_objects(Bucket=bucket, Prefix=current_endpoint_capture_prefix)
capture_files = [capture_file.get("Key") for capture_file in result.get('Contents')]
print("Found Capture Files:")
print("\n ".join(capture_files))

```

The format of the Amazon S3 path is as follows:

```

s3://{{destination-bucket-prefix}}/{{endpoint-name}}/{{variant-name}}/yyyy/mm/dd/hh/
filename.jsonl

```

## Schedule Monitoring Jobs

Amazon SageMaker Model Monitor provides you the ability to continuously monitor the data collected from the endpoints on a schedule. You can create a monitoring schedule with the [CreateMonitoringSchedule](#) API with a predefined periodic interval. For example, every x hours (x can range from 1 to 23).

With a monitoring schedule, SageMaker can kick off processing jobs at a specified frequency to analyze the data collected during a given period. SageMaker provides a prebuilt container for performing analysis on tabular datasets. In the processing job, SageMaker compares the dataset for the current analysis with the baseline statistics, constraints provided and generate a violations report. In addition, CloudWatch metrics are emitted for each feature under analysis. Alternatively, you could choose to bring your own container as outlined in the [Bring Your Own Containers \(p. 1993\)](#) topic.

You can create a model monitoring schedule for the endpoint created earlier. Use the baseline resources (constraints and statistics) to compare against the real-time traffic. For this example, upload the training dataset that was used to train the pretrained model included in this example. If you already have it in Amazon S3, you can point to it directly.

```

# copy over the training dataset to Amazon S3 (if you already have it in Amazon S3, you
# could reuse it)
baseline_prefix = prefix + '/baselining'
baseline_data_prefix = baseline_prefix + '/data'
baseline_results_prefix = baseline_prefix + '/results'

baseline_data_uri = 's3://{{}}/{{}}'.format(bucket,baseline_data_prefix)
baseline_results_uri = 's3://{{}}/{{}}'.format(bucket, baseline_results_prefix)
print('Baseline data uri: {}'.format(baseline_data_uri))
print('Baseline results uri: {}'.format(baseline_results_uri))

```

```

training_data_file = open("test_data/training-dataset-with-header.csv", 'rb')
s3_key = os.path.join(baseline_prefix, 'data', 'training-dataset-with-header.csv')
boto3.Session().resource('s3').Bucket(bucket).Object(s3_key).upload_fileobj(training_data_file)

```

Create a model monitoring schedule for the endpoint using the baseline constraints and statistics to compare against real-time traffic.

```

from sagemaker.model_monitor import CronExpressionGenerator

```

```
from time import gmtime, strftime

mon_schedule_name = 'DEMO-xgb-churn-pred-model-monitor-schedule-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
my_default_monitor.create_monitoring_schedule(
    schedule_name=mon_schedule_name,
    endpoint_input=predictor.endpoint,
    post_analytics_processor_script=s3_code_postprocessor_uri,
    output_s3_uri=s3_report_path,
    statistics=my_default_monitor.baseline_statistics(),
    constraints=my_default_monitor.suggested_constraints(),
    schedule_cron_expression=CronExpressionGenerator.hourly(),
    enable_cloudwatch_metrics=True,
)
```

Describe and inspect the schedule: After you describe it, observe that the `MonitoringScheduleStatus` in [MonitoringScheduleSummary](#) returned by the [ListMonitoringSchedules](#) API changes to `Scheduled`.

```
desc_schedule_result = my_default_monitor.describe_schedule()
print('Schedule status: {}'.format(desc_schedule_result['MonitoringScheduleStatus']))
```

## The cron Expression for Monitoring Schedule

To provide details for the monitoring schedule, use `ScheduleConfig`, which is a cron expression that describes details about the monitoring schedule.

Amazon SageMaker Model Monitor supports the following cron expressions:

- To set the job to start every hour, use the following:  
`Hourly: cron(0 * ? * * *)`
- To run the job daily, use the following:  
`cron(0 [00-23] ? * * *)`

For example, the following are valid cron expressions:

- Daily at 12 PM UTC: `cron(0 12 ? * * *)`
- Daily at 12 AM UTC: `cron(0 0 ? * * *)`

To support running every 6, 12 hours, Model Monitor supports the following expression:

```
cron(0 [00-23]/[01-24] ? * * *)
```

For example, the following are valid cron expressions:

- Every 12 hours, starting at 5 PM UTC: `cron(0 17/12 ? * * *)`
- Every two hours, starting at 12 AM UTC: `cron(0 0/2 ? * * *)`

### Notes

- Although the cron expression is set to start at 5 PM UTC, note that there could be a delay of 0-20 minutes from the actual requested time to run the execution.
- If you want to run on a daily schedule, don't provide this parameter. SageMaker picks a time to run every day.

- Currently, SageMaker only supports hourly integer rates between 1 hour and 24 hours.

## Amazon SageMaker Model Monitor Prebuilt Container

SageMaker provides a built-in container `sagemaker-model-monitor-analyzer` that provides you with a range of model monitoring capabilities, including constraint suggestion, statistics generation, constraint validation against a baseline, and emitting Amazon CloudWatch metrics. This container is based on Spark and is built with [Deequ](#). The prebuilt container for SageMaker Model Monitor can be accessed as follows:

`<ACCOUNT_ID>.dkr.ecr.<REGION_NAME>.amazonaws.com/sagemaker-model-monitor-analyzer`

For example: `159807026194.dkr.ecr.us-west-2.amazonaws.com/sagemaker-model-monitor-analyzer`

If you are in an Amazon region in China, the prebuilt container for SageMaker Model Monitor can be accessed as follows:

`<ACCOUNT_ID>.dkr.ecr.<REGION_NAME>.amazonaws.com.cn/sagemaker-model-monitor-analyzer`

The following table lists the supported values for account IDs and corresponding Amazon Region names.

ACCOUNT_ID	REGION_NAME
156813124566	us-east-1
777275614652	us-east-2
890145073186	us-west-1
159807026194	us-west-2
875698925577	af-south-1
001633400207	ap-east-1
574779866223	ap-northeast-1
709848358524	ap-northeast-2
126357580389	ap-south-1
245545462676	ap-southeast-1
563025443158	ap-southeast-2
536280801234	ca-central-1
453000072557	cn-north-1
453252182341	cn-northwest-1
048819808253	eu-central-1
895015795356	eu-north-1
933208885752	eu-south-1

ACCOUNT_ID	REGION_NAME
468650794304	eu-west-1
749857270468	eu-west-2
680080141114	eu-west-3
607024016150	me-south-1
539772159869	sa-east-1
362178532790	us-gov-west-1

To write your own analysis container, see the container contract described in [Customize Monitoring \(p. 1990\)](#).

## Interpret Results

After you run a baseline processing job and obtained statistics and constraint for your dataset, you can execute monitoring jobs that calculate statistics and list any violations encountered relative to the baseline constraints. Amazon CloudWatch metrics are also reported in your account by default. For information on viewing the results of monitoring in Amazon SageMaker Studio, see [Visualize Results in Amazon SageMaker Studio \(p. 1984\)](#).

### List Executions

The schedule starts monitoring jobs at the specified intervals. The following code lists the latest five executions. If you are running this code after creating the hourly schedule, the executions might be empty, and you might have to wait until you cross the hour boundary (in UTC) to see the executions start. The following code includes the logic for waiting.

```
mon_executions = my_default_monitor.list_executions()
print("We created a hourly schedule above and it will kick off executions ON the hour (plus
0 - 20 min buffer.\nWe will have to wait till we hit the hour...")

while len(mon_executions) == 0:
    print("Waiting for the 1st execution to happen...")
    time.sleep(60)
    mon_executions = my_default_monitor.list_executions()
```

### Inspect a Specific Execution

In the previous step, you picked up the latest completed or failed scheduled execution. You can explore what went right or wrong. The terminal states are:

- **Completed** – The monitoring execution completed and no issues were found in the violations report.
- **CompletedWithViolations** – The execution completed, but constraint violations were detected.
- **Failed** – The monitoring execution failed, possibly due to client error (for example, a role issues) or infrastructure issues. To identify the cause, see the `FailureReason` and `ExitMessage`.

```
latest_execution = mon_executions[-1] # latest execution's index is -1, previous is -2 and
so on..
time.sleep(60)
latest_execution.wait(logs=False)
```

```

print("Latest execution status: {}".format(latest_execution.describe()
['ProcessingJobStatus']))
print("Latest execution result: {}".format(latest_execution.describe()['ExitMessage']))

latest_job = latest_execution.describe()
if (latest_job['ProcessingJobStatus'] != 'Completed'):
    print("====STOP==== \n No completed executions to inspect further. Please wait till
an execution completes or investigate previously reported failures.")

```

```

report_uri=latest_execution.output.destination
print('Report Uri: {}'.format(report_uri))

```

## List Generated Reports

List the generated reportsUse the following code to list the generated reports.

```

from urllib.parse import urlparse
s3uri = urlparse(report_uri)
report_bucket = s3uri.netloc
report_key = s3uri.path.lstrip('/')
print('Report bucket: {}'.format(report_bucket))
print('Report key: {}'.format(report_key))

s3_client = boto3.Session().client('s3')
result = s3_client.list_objects(Bucket=report_bucket, Prefix=report_key)
report_files = [report_file.get("Key") for report_file in result.get('Contents')]
print("Found Report Files:")
print("\n ".join(report_files))

```

## Violations Report

If there are violations compared to the baseline, they are generated in the violations report. Use the following code to list the violations.

```

violations = my_default_monitor.latest_monitoring_constraint_violations()
pd.set_option('display.max_colwidth', -1)
constraints_df = pd.io.json.json_normalize(violations.body_dict["violations"])
constraints_df.head(10)

```

This applies only to datasets that contain tabular data. The following schema files specify the statistics calculated and the violations monitored for.

### Output Files for Tabular Datasets

File Name	Description
<b>statistics.json</b>	Contains columnar statistics for each feature in the dataset that is analyzed. See the schema of this file in the next topic.  <b>Note</b> This file is created only for data quality monitoring.
<b>constraintsViolations.json</b>	Contains a list of violations found in this current set of data as compared to the baseline statistics and constraints file

File Name	Description
	specified in the <code>baseline_constraints</code> and <code>baseline_statistics</code> paths.

The [Amazon SageMaker Model Monitor Prebuilt Container \(p. 1981\)](#) saves a set of Amazon CloudWatch metrics for each feature by default.

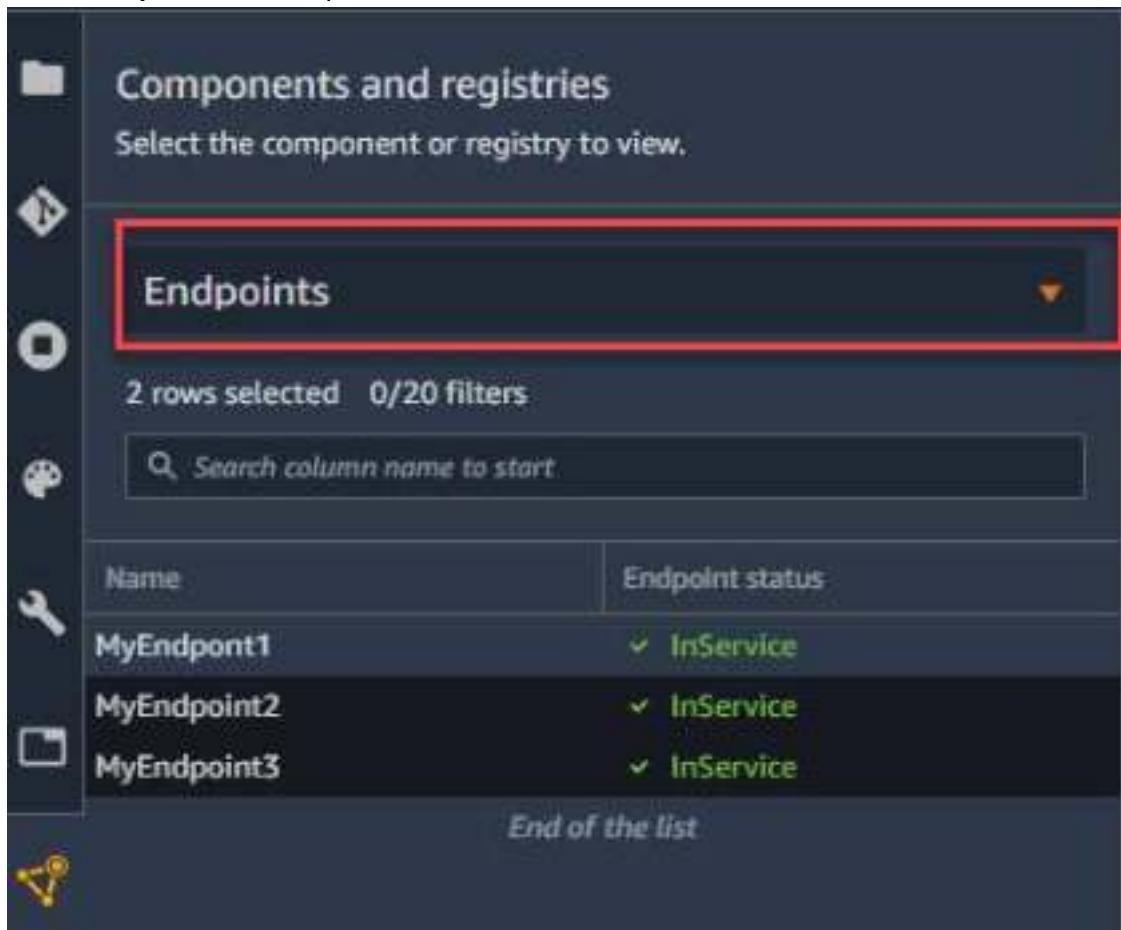
The container code can emit CloudWatch metrics in this location: `/opt/ml/output/metrics/cloudwatch`.

## Visualize Results in Amazon SageMaker Studio

You can also visualize the results of monitoring in Amazon SageMaker Studio. You can view the details of any monitoring job run, and you can create charts that show the baseline and captured values for any metric that the monitoring job calculates.

### To view the detailed results of a monitoring job

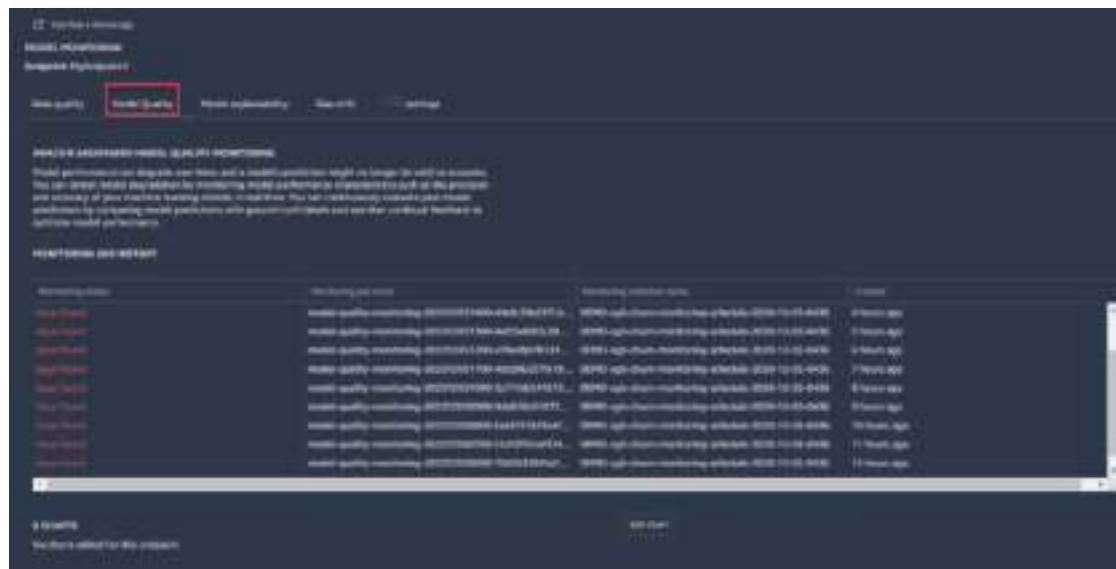
1. Sign in to Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
2. In the left navigation pane, choose the **Components and registries** icon (- 3. Choose **Endpoints** in the drop-down menu.



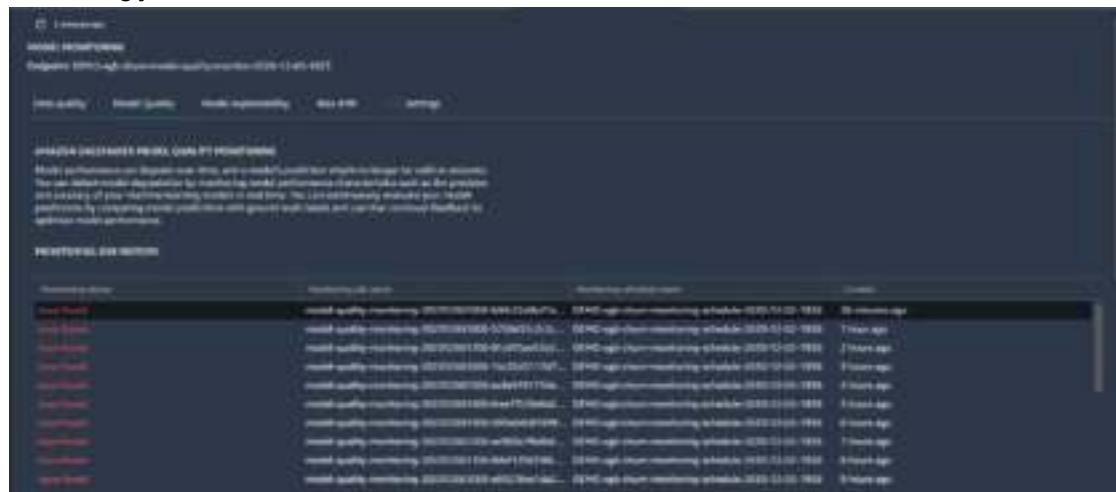
The screenshot shows the 'Components and registries' page in Amazon SageMaker Studio. On the left, there is a vertical sidebar with icons for different components: a folder for 'Components and registries', a diamond for 'Data Wrangler', a square for 'ML Projects', a paint palette for 'Notebooks', a wrench for 'Endpoints', a folder for 'Transforms', and a key for 'Monitoring'. The main area has a dark background with light-colored text. At the top, it says 'Components and registries' and 'Select the component or registry to view.' Below that, a dropdown menu is open, with the option 'Endpoints' highlighted by a red rectangle. Underneath the dropdown, it says '2 rows selected 0/20 filters' and there is a search bar with the placeholder 'Search column name to start'. A table follows, with columns 'Name' and 'Endpoint status'. It contains three rows: 'MyEndpoint1' with 'InService', 'MyEndpoint2' with 'InService', and 'MyEndpoint3' with 'InService'. At the bottom of the table, it says 'End of the list'. The entire interface is framed by a thick red border.

Name	Endpoint status
MyEndpoint1	✓ InService
MyEndpoint2	✓ InService
MyEndpoint3	✓ InService

4. On the endpoint tab, choose the monitoring type for which you want to see job details.



- Choose the name of the monitoring job run for which you want to view details from the list of monitoring jobs.



6. The **MONITORING JOB DETAILS** tab opens with a detailed report of the monitoring job.

**MONITORING JOB DETAILS**

Monitoring Execution Name: model-quality-monitoring-202012061900-h0455d6a21a4e972867608

Processing Job ARN: arn:aws:sagemaker-us-east-2:123456789012:processing-job/model-quality-monitoring-202012061900-h0455d6a21a4e972867608

Monitoring Schedule: DEMO-rgb-churn-monitoring-schedule-2020-12-02-1958

Monitoring Job Status: Completed With Violations

**MONITORING JOB REPORT**

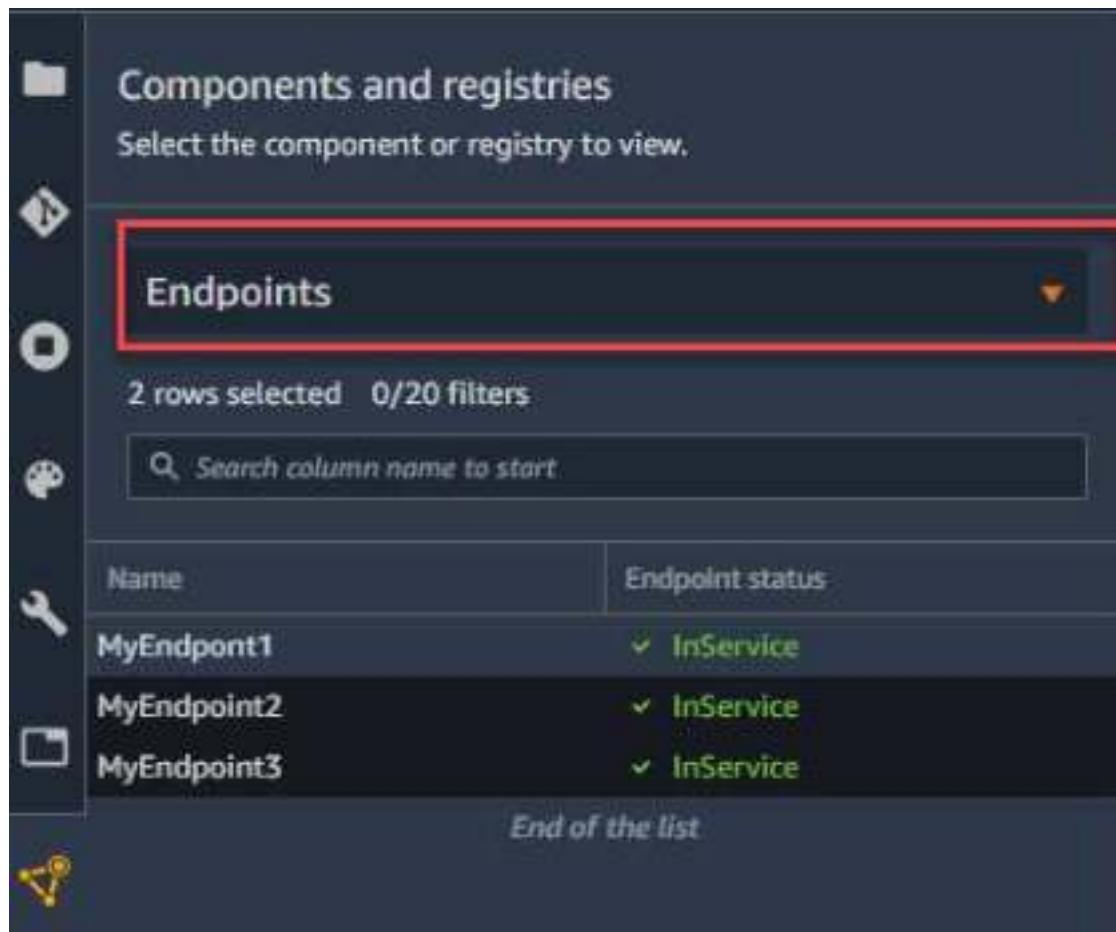
Amazon SageMaker Model Monitor compared this run against the baseline and detected these constraint violations.

Constraint	Violation details
LessThanThreshold	Metric precision with 0.7684444444444445 +/- 0.00601752812981426 was LessThanThreshold '1.0'
LessThanThreshold	Metric truePositiveRate with 0.0648480373105345 +/- 0.00163285764989067 was LessThanThreshold '0.5714285714285714'
LessThanThreshold	Metric f1 with 0.12294456958620442 +/- 0.0027741665172884867 was LessThanThreshold '0.7272727272727273'
LessThanThreshold	Metric accuracy with 0.30909876265466815 +/- 0.0011167989498387929 was LessThanThreshold '0.9402985074636860'
Greater Than Threshold	Metric falsePositiveRate with 0.05395658189216684 +/- 0.0018372499707814655 was GreaterThanThreshold '0.0'
LessThanThreshold	Metric trueNegativeRate with 0.9460854189079331 +/- 0.0018372499707814401 was LessThanThreshold '1.0'
Greater Than Threshold	Metric falseNegativeRate with 0.3331519626894679 +/- 0.001632857649890645 was GreaterThanThreshold '0.4285714285714286'
LessThanThreshold	Metric recall with 0.0668480373105345 +/- 0.00163285764989067 was LessThanThreshold '0.5714285714285714'
LessThanThreshold	Metric f2 with 0.08177239854615355 +/- 0.0019556109564544965 was LessThanThreshold '0.623'

You can create a chart that displays the baseline and captured metrics for a time period.

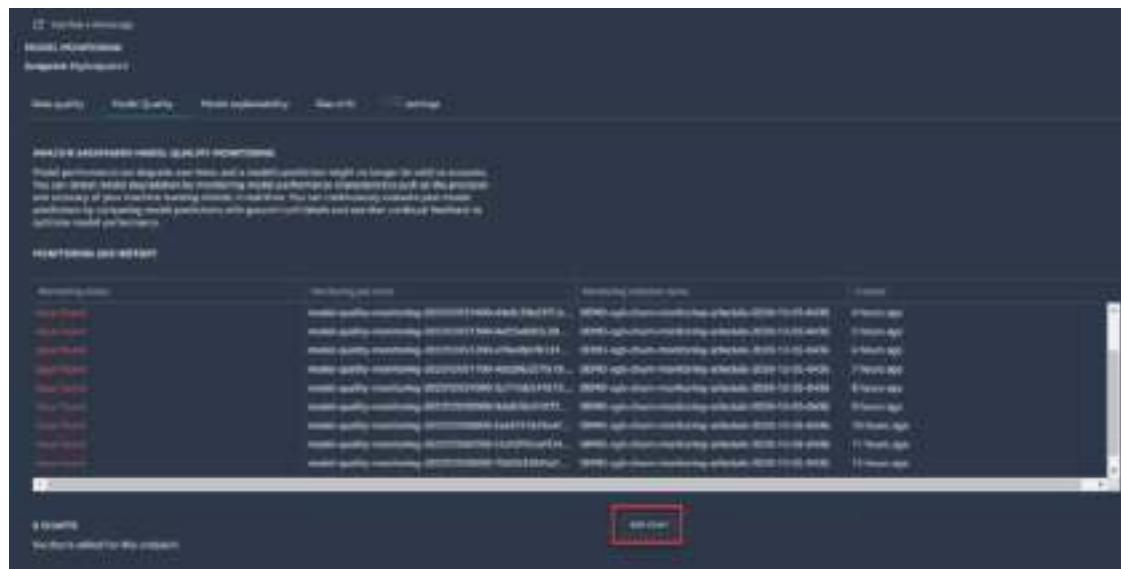
### To create a chart in SageMaker Studio to visualize monitoring results

1. Sign in to Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
2. In the left navigation pane, choose the **Components and registries** icon (  ).
3. Choose **Endpoints** in the drop-down menu.

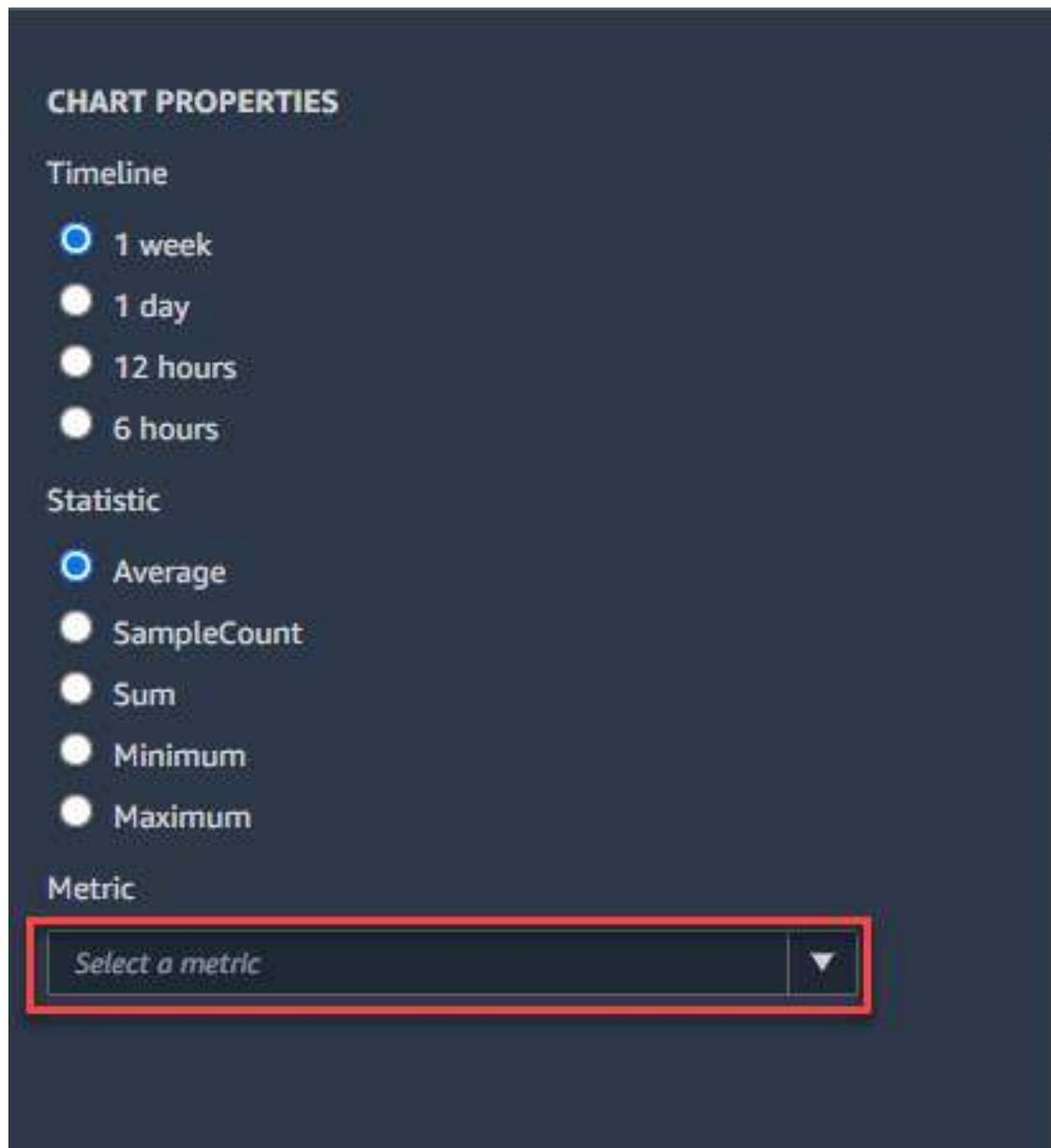


- On the **Endpoint** tab, choose the monitoring type you want to create a chart for. This example shows a chart for the **Model quality** monitoring type.

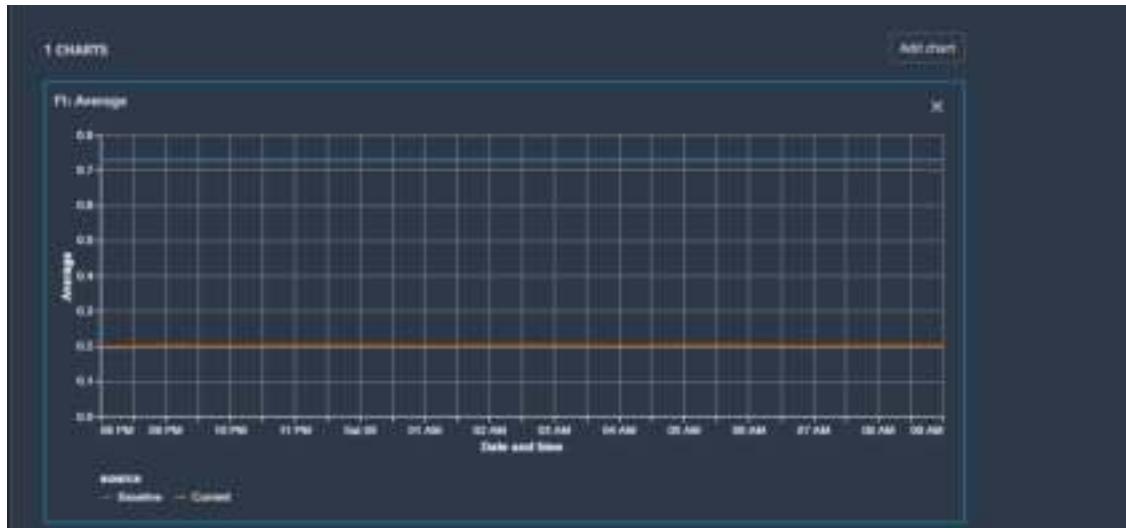
- ## 5. Choose Add chart.



6. On the **CHART PROPERTIES** tab, choose the time period, statistic, and metric that you want to chart. This example shows a chart for a **Timeline of 1 week**, the **Average Statistic** of, and the **F1 Metric**.



7. The chart that shows the baseline and current metric statistic you chose in the previous step shows up in the **Endpoint** tab.



## Advanced Topics

The following sections contain more advanced tasks that explain how to customize monitoring using preprocessing and postprocessing scripts, how to build your own container, and how to use Amazon CloudFormation to create a monitoring schedule.

### Topics

- [Customize Monitoring \(p. 1990\)](#)
- [Create a Monitoring Schedule with an Amazon CloudFormation Custom Resource \(p. 2000\)](#)

## Customize Monitoring

In addition to using the built-in monitoring mechanisms, you can create your own custom monitoring schedules and procedures using preprocessing and postprocessing scripts or by using or building your own container.

### Topics

- [Preprocessing and Postprocessing \(p. 1990\)](#)
- [Bring Your Own Containers \(p. 1993\)](#)

## Preprocessing and Postprocessing

In addition to using the built-in mechanisms, you can extend the code with the preprocessing and postprocessing scripts.

### Topics

- [Postprocessing Script \(p. 1990\)](#)
- [Preprocessing Script \(p. 1991\)](#)

## Postprocessing Script

You can extend the code with the postprocessing script by following this contract:

```
def postprocess_handler():
    print("Hello from post-proc script!")
```

Specify it as a path in Amazon Simple Storage Service (Amazon S3) in the CreateMonitoringSchedule request, as shown following:

```
.MonitoringAppSpecification.PostAnalyticsProcessorSourceUri.
```

### Preprocessing Script

The Amazon SageMaker Model Monitor container works only with tabular or flattened JSON structures. We provide a per-record preprocessor for some small changes required to transform the dataset. For example, if your output is an array [1.0, 2.1], you need to convert this into a flattened JSON, like {"prediction0": 1.0, "prediction1" : 2.1"}. A sample implementation might look like the following:

```
def preprocess_handler(inference_record):

    input_data = inference_record.endpoint_input.data
    output_data = inference_record.endpoint_output.data

    input_data['feature0'] = random.randint(1, 3)
    input_data['feature1'] = random.uniform(0, 1.6)
    input_data['feature2'] = random.uniform(0, 1.6)

    output_data['prediction0'] = random.uniform(1, 30)

    return {**input_data, **output_data}
```

Specify it as a path in Amazon S3 in the CreateMonitoringSchedule request:

```
.MonitoringAppSpecification.RecordPreprocessorSourceUri.
```

The structure of the inference\_record is defined as follows:

```
KEY_EVENT_METADATA = "eventMetadata"
KEY_EVENT_METADATA_EVENT_ID = "eventId"
KEY_EVENT_METADATA_EVENT_TIME = "inferenceTime"
KEY_EVENT_METADATA_CUSTOM_ATTR = "customAttributes"

KEY_EVENTDATA = "captureData"
KEY_EVENTDATA_INPUT = "endpointInput"
KEY_EVENTDATA_OUTPUT = "endpointOutput"
KEY_EVENTDATA_ENCODING = "encoding"
KEY_EVENTDATA_DATA = "data"
KEY_EVENTDATA_OBSERVED_CONTENT_TYPE = "observedContentType"
KEY_EVENTDATA_MODE = "mode"

KEY_EVENT_VERSION = "eventVersion"

"""
{
    "captureData": {
        "endpointInput": {
            "observedContentType": "text/csv",
            "mode": "INPUT",
            "data": "132,25,113.2,96,269.9,107,,0,0,0,0,0,0,1,0,1,0,0,1",
            "encoding": "CSV"
        },
        "endpointOutput": {
            "observedContentType": "text/csv; charset=utf-8",
```

```

        "mode": "OUTPUT",
        "data": "0.01076381653547287",
        "encoding": "CSV"
    }
},
"eventMetadata": {
    "eventId": "fecalab1-8025-47e3-8f6a-99e3fdd7b8d9",
    "inferenceTime": "2019-11-20T23:33:12Z"
},
"eventVersion": "0"
}
"""

class EventConfig:
    def __init__(self, endpoint, variant, start_time, end_time):
        self.endpoint = endpoint
        self.variant = variant
        self.start_time = start_time
        self.end_time = end_time

class EventMetadata:
    def __init__(self, event_metadata_dict):
        self.event_id = event_metadata_dict.get(KEY_EVENT_METADATA_EVENT_ID, None)
        self.event_time = event_metadata_dict.get(KEY_EVENT_METADATA_EVENT_TIME, None)
        self.custom_attribute =
event_metadata_dict.get(KEY_EVENTDATA_OBSERVED_CONTENT_TYPE, None)

class EventData:
    def __init__(self, data_dict):
        self.encoding = data_dict.get(KEY_EVENTDATA_ENCODING, None)
        self.data = data_dict.get(KEY_EVENTDATA_DATA, None)
        self.observedContentType = data_dict.get(KEY_EVENTDATA_OBSERVED_CONTENT_TYPE, None)
        self.mode = data_dict.get(KEY_EVENTDATA_MODE, None)

    def as_dict(self):
        ret = {
            KEY_EVENTDATA_ENCODING: self.encoding,
            KEY_EVENTDATA_DATA: self.data,
            KEY_EVENTDATA_OBSERVED_CONTENT_TYPE: self.observedContentType,
        }
        return ret

class CapturedData:
    def __init__(self, event_dict):
        self.event_metadata = None
        self.endpoint_input = None
        self.endpoint_output = None
        self.event_version = None
        self.event_dict = event_dict
        self._event_dict_postprocessed = False
        if KEY_EVENT_METADATA in event_dict:
            self.event_metadata = EventMetadata(event_dict[KEY_EVENT_METADATA])
        if KEY_EVENTDATA in event_dict:
            if KEY_EVENTDATA_INPUT in event_dict[KEY_EVENTDATA]:
                self.endpoint_input = EventData(event_dict[KEY_EVENTDATA][KEY_EVENTDATA_INPUT])
            if KEY_EVENTDATA_OUTPUT in event_dict[KEY_EVENTDATA]:
                self.endpoint_output = EventData(event_dict[KEY_EVENTDATA][KEY_EVENTDATA_OUTPUT])
            if KEY_EVENT_VERSION in event_dict:
                self.event_version = event_dict[KEY_EVENT_VERSION]

    def as_dict(self):

```

```
if self._event_dict_postprocessed is True:
    return self.event_dict
if KEY_EVENTDATA in self.event_dict:
    if KEY_EVENTDATA_INPUT in self.event_dict[KEY_EVENTDATA]:
        self.event_dict[KEY_EVENTDATA][KEY_EVENTDATA_INPUT] =
self.endpoint_input.as_dict()
    if KEY_EVENTDATA_OUTPUT in self.event_dict[KEY_EVENTDATA]:
        self.event_dict[KEY_EVENTDATA][
            KEY_EVENTDATA_OUTPUT
        ] = self.endpoint_output.as_dict()
self._event_dict_postprocessed = True
return self.event_dict
```

## Bring Your Own Containers

Amazon SageMaker Model Monitor provides a prebuilt container with ability to analyze the data captured from endpoints for tabular datasets. If you would like to bring your own container, Model Monitor provides extension points which you can leverage.

Under the hood, when you create a `MonitoringSchedule`, Model Monitor ultimately kicks off processing jobs. Hence the container needs to be aware of the processing job contract documented in the [Build Your Own Processing Container \(Advanced Scenario\) \(p. 668\)](#) topic. Note that Model Monitor kicks off the processing job on your behalf per the schedule. While invoking, Model Monitor sets up additional environment variables for you so that your container has enough context to process the data for that particular execution of the scheduled monitoring. For additional information on container inputs, see the [Container Contract Inputs \(p. 1993\)](#).

In the container, using the above environment variables/context, you can now analyze the dataset for the current period in your custom code. After this analysis is complete, you can chose to emit your reports to be uploaded to an S3 bucket. The reports that the prebuilt container generates are documented in [Container Contract Outputs \(p. 1995\)](#). If you would like the visualization of the reports to work in SageMaker Studio, you should follow the same format. You can also choose to emit completely custom reports.

You also emit CloudWatch metrics from the container by following the instructions in [CloudWatch Metrics for Bring Your Own Containers \(p. 1999\)](#).

### Topics

- [Container Contract Inputs \(p. 1993\)](#)
- [Container Contract Outputs \(p. 1995\)](#)
- [CloudWatch Metrics for Bring Your Own Containers \(p. 1999\)](#)

### Container Contract Inputs

The Amazon SageMaker Model Monitor platform invokes your container code according to a specified schedule. If you chose to write your own container code, the following environment variables are available for your container code. In this context, you can analyze the current dataset or evaluate the constraints if you chose to and emit metrics, if applicable.

```
"Environment": {
    "dataset_format": "{\"sagemakerCaptureJson\": {\"captureIndexNames\": [\"endpointInput\",
\"endpointOutput\"]}}",
    "dataset_source": "/opt/ml/processing/endpointdata",
    "end_time": "2019-12-01T16: 20: 00Z",
    "output_path": "/opt/ml/processing/resultdata",
    "publish_cloudwatch_metrics": "Disabled",
    "sagemaker_endpoint_name": "endpoint-name",
    "sagemaker_monitoring_schedule_name": "schedule-name",
    "start_time": "2019-12-01T15: 20: 00Z"
```

}

## Parameters

Parameter Name	Description
<code>dataset_format</code>	For a job started from a <code>MonitoringSchedule</code> backed by an <code>Endpoint</code> , this is <code>sageMakerCaptureJson</code> with the capture indices <code>endpointInput</code> , or <code>endpointOutput</code> , or both.
<code>dataset_source</code>	The local path in which the data corresponding to the monitoring period, as specified by <code>start_time</code> and <code>end_time</code> , are available. At this path, the data is available in <code>/{endpoint-name}/{variant-name}/yyyy/mm/dd/hh</code> .  We sometimes download more than what is specified by the start and end times. It is up to the container code to parse the data as required.
<code>output_path</code>	The local path to write output reports and other files. You specify this parameter in the <code>CreateMonitoringSchedule</code> request as <code>MonitoringOutputConfig.MonitoringOutput[0].LocalPath</code> . It is uploaded to the S3Uri path specified in <code>MonitoringOutputConfig.MonitoringOutput[0].S3Uri</code> .
<code>publish_cloudwatch_metrics</code>	For a job launched by <code>CreateMonitoringSchedule</code> , this parameter is set to <code>Enabled</code> . The container can choose to write the Amazon CloudWatch output file at <code>[filepath]</code> .
<code>sagemaker_endpoint_name</code>	The name of the <code>Endpoint</code> that this scheduled job was launched for.
<code>sagemaker_monitoring_schedule_name</code>	The name of the <code>MonitoringSchedule</code> that launched this job.
<code>*sagemaker_endpoint_datacapture_prefix*</code>	The prefix specified in the <code>DataCaptureConfig</code> parameter of the <code>Endpoint</code> . The container can use this if it needs to directly access more data than already downloaded by SageMaker at the <code>dataset_source</code> path.
<code>start_time, end_time</code>	The time window for this analysis run. For example, for a job scheduled to run at 05:00 UTC and a job that runs on 20/02/2020, <code>start_time:</code> is <code>2020-02-19T06:00:00Z</code> and <code>end_time:</code> is <code>2020-02-20T05:00:00Z</code>
<code>baseline_constraints:</code>	The local path of the baseline constraint file specified in <code>BaselineConfig.ConstraintResource.S3Uri</code> . This is available only if this parameter was specified in the <code>CreateMonitoringSchedule</code> request.

Parameter Name	Description
<code>baseline_statistics</code>	The local path to the baseline statistics file specified in <code>BaselineConfig.StatisticsResource.S3Uri</code> . This is available only if this parameter was specified in the <code>CreateMonitoringSchedule</code> request.

### Container Contract Outputs

The container can analyze the data available in the `*dataset_source*` path and write reports to the path in `*output_path*`. The container code can write any reports that suit your needs.

If you use the following structure and contract, certain output files are treated specially by SageMaker in the visualization and API . This applies only to tabular datasets.

### Output Files for Tabular Datasets

File Name	Description
<code>statistics.json</code>	This file is expected to have columnar statistics for each feature in the dataset that is analyzed. The schema for this file is available in the next section.
<code>constraints.json</code>	This file is expected to have the constraints on the features observed. The schema for this file is available in the next section.
<code>constraintsViolations.json</code>	This file is expected to have the list of violations found in this current set of data as compared to the baseline statistics and constraints file specified in the <code>baseline_constraints</code> and <code>baseline_statistics</code> path.

In addition, if the `publish_cloudwatch_metrics` value is "Enabled" container code can emit Amazon CloudWatch metrics in this location: `/opt/ml/output/metrics/cloudwatch`. The schema for these files is described in the following sections.

#### Topics

- [Schema for Statistics \(statistics.json file\) \(p. 1995\)](#)
- [Schema for Constraints \(constraints.json file\) \(p. 1997\)](#)

#### Schema for Statistics (statistics.json file)

The schema defined in the `statistics.json` file specifies the statistical parameters to be calculated for the baseline and data that is captured. It also configures the bucket to be used by `KLL`, a very compact quantiles sketch with lazy compaction scheme.

```
{
    "version": 0,
    # dataset level stats
    "dataset": {
        "item_count": number
    },
    # feature level stats
```

```

"features": [
  {
    "name": "feature-name",
    "inferred_type": "Fractional" | "Integral",
    "numerical_statistics": {
      "common": {
        "num_present": number,
        "num_missing": number
      },
      "mean": number,
      "sum": number,
      "std_dev": number,
      "min": number,
      "max": number,
      "distribution": {
        "kll": {
          "buckets": [
            {
              "lower_bound": number,
              "upper_bound": number,
              "count": number
            }
          ],
          "sketch": {
            "parameters": {
              "c": number,
              "k": number
            },
            "data": [
              [
                num,
                num,
                num,
                num
              ],
              [
                num,
                num
              ][
                num,
                num
              ]
            ]
          }
        }#sketch
      }#KLL
    }#distribution
  }#num_stats
},
{
  "name": "feature-name",
  "inferred_type": "String",
  "string_statistics": {
    "common": {
      "num_present": number,
      "num_missing": number
    },
    "distinct_count": number,
    "distribution": {
      "categorical": {
        "buckets": [
          {
            "value": "string",
            "count": number
          }
        ]
      }
    }
  }
}

```

```

        }
    },
    #provision for custom stats
]
}
}
```

## Notes

- The specified metrics are recognized by SageMaker in later visualization changes. The container can emit more metrics if required.
- [KLL sketch](#) is the recognized sketch. Custom containers can write their own representation, but it won't be recognized by SageMaker in visualizations.
- By default, the distribution is materialized in 10 buckets. You can't change this.

## Schema for Constraints (constraints.json file)

A constraints.json file is used to express the constraints that a dataset must satisfy. Amazon SageMaker Model Monitor containers can use the constraints.json file to evaluate datasets against. Prebuilt containers provide the ability to generate the constraints.json file automatically for a baseline dataset. If you bring your own container, you can provide it with similar abilities or you can create the constraints.json file in some other way. Here is the schema for the constraint file that the prebuilt container uses. Bring your own containers can adopt the same format or enhance it as required.

```
{
  "version" : 0,
  "features": [
    {
      "name": "string",
      "inferred_type": "Integral" | "Fractional" |
        | "String" | "Unknown",
      "completeness": number, # denotes observed non-null value percentage
      "num_constraints" : {
        "is_non_negative": boolean,
      },
      "string_constraints" : {
        "domains": [
          "list of",
          "observed values",
          "for small cardinality"
        ],
      },
      "monitoringConfigOverrides" : {
        #monitoringConfigOverrides
      }#feature
    }#features

    # options to control monitoring for this feature with monitoring jobs
    # See the following table for notes on what each constraint is doing.
  "monitoring_config": {
    "evaluate_constraints": "Enabled",
    "emit_metrics": "Enabled",
    "datatype_check_threshold": 0.1,
    "domain_content_threshold": 0.1,
    "distribution_constraints": {
      "perform_comparison": "Enabled",
      "comparison_threshold": 0.1,
      "comparison_method": "Simple" || "Robust"
    }
  }
}
```

```
}}#schema
```

## Monitoring Constraints

Constraint	Description
<code>evaluate_constraints</code>	<p>When <code>Enabled</code>, evaluates whether the current dataset being analyzed satisfies the constraints specified in the <code>constraints.json</code> file taken as a baseline.</p> <p>Valid values: <code>Enabled</code> or <code>Disabled</code></p> <p>Default: <code>Enabled</code></p>
<code>emit_metrics</code>	<p>When <code>Enabled</code>, emits CloudWatch metrics for the data contained in the file.</p> <p>Valid values: <code>Enabled</code> or <code>Disabled</code></p> <p>Default: <code>Enabled</code></p>
<code>datatype_check_threshold</code>	<p>If the threshold is above the value of the specified <code>datatype_check_threshold</code>, this causes a failure that is treated as a violation in the violation report. If the data types in the current execution are not the same as in the baseline dataset, this threshold is used to evaluate if it needs to be flagged as a violation.</p> <p>During the baseline step, the generated constraints suggest the inferred data type for each column. The <code>datatype_check_threshold</code> parameter can be tuned to adjust the threshold on when it is flagged as a violation.</p> <p>Valid values: float</p> <p>Default: 0.1</p>
<code>domain_content_threshold</code>	<p>If there are more unknown values for a String field in the current dataset than in the baseline dataset, this threshold can be used to dictate if it needs to be flagged as a violation.</p> <p>Valid values: float</p> <p>Default: 0.1</p>
<code>distribution_constraints</code>	<p><code>perform_comparison</code></p> <p>When <code>Enabled</code>, this flag instructs the code to perform a distribution comparison between the baseline distribution and the distribution observed for the current dataset.</p> <p>Valid values: <code>Enabled</code> or <code>Disabled</code></p> <p>Default: <code>Enabled</code></p>

Constraint	Description
	<p>comparison_threshold</p> <p>If the threshold is above the value set for the comparison_threshold, this causes a failure that is treated as a violation in the violation report. The distance is calculated by getting the maximum absolute difference between the cumulative distribution functions of two distributions.</p> <p>Valid values: float</p> <p>Default: 0.1</p>
	<p>comparison_method</p> <p>Whether to calculate linf_simple or linf_robust. The linf_simple is based on the maximum absolute difference between the cumulative distribution functions of two distributions. Calculating linf_robust is based on linf_simple, but is used when there are not enough samples. The linf_robust formula is based on the <a href="#">Two-sample Kolmogorov–Smirnov test</a>.</p> <p>Valid values: linf_simple or linf_robust.</p>

### CloudWatch Metrics for Bring Your Own Containers

If the publish\_cloudwatch\_metrics value is Enabled in the Environment map in the /opt/ml/processing/processingjobconfig.json file, the container code emits Amazon CloudWatch metrics in this location: /opt/ml/output/metrics/cloudwatch.

The schema for this file is closely based on the CloudWatch PutMetrics API. The namespace is not specified here. It defaults to /aws/sagemaker/Endpoint/data-metrics. However, you can specify dimensions. We recommend that you add the Endpoint and MonitoringSchedule dimensions at a minimum.

```
{
    "MetricName": "", # Required
    "Timestamp": "2019-11-26T03:00:00Z", # Required
    "Dimensions" : [{"Name": "Endpoint", "Value": "endpoint_0"}, {"Name": "MonitoringSchedule", "Value": "schedule_0"}]
    "Value": Float,
    # Either the Value or the StatisticValues field can be populated and not both.
    "StatisticValues": {
        "SampleCount": Float,
        "Sum": Float,
        "Minimum": Float,
        "Maximum": Float
    },
    "Unit": "Count", # Optional
}
```

## Create a Monitoring Schedule with an Amazon CloudFormation Custom Resource

To use Amazon CloudFormation to create a monitoring schedule, use an Amazon CloudFormation custom resource. The custom resource is in Python. To deploy it, see [Python Lambda deployment](#).

### Custom Resource

Start by adding a custom resource to your Amazon CloudFormation template. This points to a Amazon Lambda function that you create in the next step.

This resource enables you to customize the parameters for the monitoring schedule. You can add or remove more parameters by modifying the Amazon CloudFormation resource and the Lambda function in the following example resource.

```
{  
    "AWSTemplateFormatVersion": "2010-09-09",  
    "Resources": {  
        "MonitoringSchedule": {  
            "Type": "Custom::MonitoringSchedule",  
            "Version": "1.0",  
            "Properties": {  
                "ServiceToken": "arn:aws:lambda:us-west-2:111111111111:function:lambda-  
name",  
                "ScheduleName": "YourScheduleName",  
                "EndpointName": "YourEndpointName",  
                "BaselineConstraintsUri": "s3://your-baseline-constraints/  
constraints.json",  
                "BaselineStatisticsUri": "s3://your-baseline-stats/statistics.json",  
                "PostAnalyticsProcessorSourceUri": "s3://your-post-processor/  
postprocessor.py",  
                "RecordPreprocessorSourceUri": "s3://your-preprocessor/preprocessor.py",  
                "InputLocalPath": "/opt/ml/processing/endpointdata",  
                "OutputLocalPath": "/opt/ml/processing/localpath",  
                "OutputS3URI": "s3://your-output-uri",  
                "ImageURI": "11111111111.dkr.ecr.us-west-2.amazonaws.com/your-image",  
                "ScheduleExpression": "cron(0 * ? * * *)",  
                "PassRoleArn": "arn:aws:iam::111111111111:role/AmazonSageMaker-  
ExecutionRole"  
            }  
        }  
    }  
}
```

### Lambda Custom Resource Code

This Amazon CloudFormation custom resource uses the [Custom Resource Helper](#) Amazon library, which you can install with pip using `pip install crhelper`.

This Lambda function is invoked by Amazon CloudFormation during the creation and deletion of the stack. This Lambda function is responsible for creating and deleting the monitoring schedule and using the parameters defined in the custom resource described in the preceding section.

```
import boto3  
import botocore  
import logging  
  
from crhelper import CfnResource  
from botocore.exceptions import ClientError
```

```
logger = logging.getLogger(__name__)
sm = boto3.client('sagemaker')

# cfnhelper makes it easier to implement a CloudFormation custom resource
helper = CfnResource()

# CFN Handlers

def handler(event, context):
    helper(event, context)

@helper.create
def create_handler(event, context):
    """
    Called when CloudFormation custom resource sends the create event
    """
    create_monitoring_schedule(event)

@helper.delete
def delete_handler(event, context):
    """
    Called when CloudFormation custom resource sends the delete event
    """
    schedule_name = get_schedule_name(event)
    delete_monitoring_schedule(schedule_name)

@helper.poll_create
def poll_create(event, context):
    """
    Return true if the resource has been created and false otherwise so
    CloudFormation polls again.
    """
    schedule_name = get_schedule_name(event)
    logger.info('Polling for creation of schedule: %s', schedule_name)
    return is_schedule_ready(schedule_name)

@helper.update
def noop():
    """
    Not currently implemented but crhelper will throw an error if it isn't added
    """
    pass

# Helper Functions

def get_schedule_name(event):
    return event['ResourceProperties']['ScheduleName']

def create_monitoring_schedule(event):
    schedule_name = get_schedule_name(event)
    monitoring_schedule_config = create_monitoring_schedule_config(event)

    logger.info('Creating monitoring schedule with name: %s', schedule_name)

    sm.create_monitoring_schedule(
        MonitoringScheduleName=schedule_name,
        MonitoringScheduleConfig=monitoring_schedule_config)

def is_schedule_ready(schedule_name):
    is_ready = False

    schedule = sm.describe_monitoring_schedule(MonitoringScheduleName=schedule_name)
    status = schedule['MonitoringScheduleStatus']
```

```

if status == 'Scheduled':
    logger.info('Monitoring schedule (%s) is ready', schedule_name)
    is_ready = True
elif status == 'Pending':
    logger.info('Monitoring schedule (%s) still creating, waiting and polling
again...', schedule_name)
else:
    raise Exception('Monitoring schedule ({}) has unexpected status:
{}'.format(schedule_name, status))

return is_ready

def create_monitoring_schedule_config(event):
    props = event['ResourceProperties']

    return {
        "ScheduleConfig": {
            "ScheduleExpression": props["ScheduleExpression"],
        },
        "MonitoringJobDefinition": {
            "BaselineConfig": {
                "ConstraintsResource": {
                    "S3Uri": props['BaselineConstraintsUri'],
                },
                "StatisticsResource": {
                    "S3Uri": props['BaselineStatisticsUri'],
                }
            },
            "MonitoringInputs": [
                {
                    "EndpointInput": {
                        "EndpointName": props["EndpointName"],
                        "LocalPath": props["InputLocalPath"],
                    }
                }
            ],
            "MonitoringOutputConfig": {
                "MonitoringOutputs": [
                    {
                        "S3Output": {
                            "S3Uri": props["OutputS3URI"],
                            "LocalPath": props["OutputLocalPath"],
                        }
                    }
                ],
            },
            "MonitoringResources": {
                "ClusterConfig": {
                    "InstanceCount": 1,
                    "InstanceType": "ml.t3.medium",
                    "VolumeSizeInGB": 50,
                }
            },
            "MonitoringAppSpecification": {
                "ImageUri": props["ImageURI"],
                "RecordPreprocessorSourceUri": props['PostAnalyticsProcessorSourceUri'],
                "PostAnalyticsProcessorSourceUri": props['PostAnalyticsProcessorSourceUri'],
                "StoppingCondition": {
                    "MaxRuntimeInSeconds": 300
                },
                "RoleArn": props["PassRoleArn"],
            }
        }
    }

```

```
def delete_monitoring_schedule(schedule_name):
    logger.info('Deleting schedule: %s', schedule_name)
    try:
        sm.delete_monitoring_schedule(MonitoringScheduleName=schedule_name)
    except ClientError as e:
        if e.response['Error']['Code'] == 'ResourceNotFoundException':
            logger.info('Resource not found, nothing to delete')
        else:
            logger.error('Unexpected error while trying to delete monitoring schedule')
            raise e
```

## Register and Deploy Models with Model Registry

With the SageMaker model registry you can do the following:

- Catalog models for production.
- Manage model versions.
- Associate metadata, such as training metrics, with a model.
- Manage the approval status of a model.
- Deploy models to production.
- Automate model deployment with CI/CD.

Catalog models by creating model package groups that contain different versions of a model. You can create a model group that tracks all of the models that you train to solve a particular problem. You can then register each model you train and the model registry adds it to the model group as a new model version. A typical workflow might look like the following:

- Create a model group.
- Create an ML pipeline that trains a model. For information about SageMaker pipelines, see [Create and Manage SageMaker Pipelines \(p. 2201\)](#).
- For each run of the ML pipeline, create a model version that you register in the model group you created in the first step.

The following topics show how to use the model registry.

### Topics

- [Create a Model Group \(p. 2003\)](#)
- [Register a Model Version \(p. 2007\)](#)
- [View Model Groups and Versions \(p. 2008\)](#)
- [View the Details of a Model Version \(p. 2010\)](#)
- [Update the Approval Status of a Model \(p. 2012\)](#)
- [Deploy a Model in the Registry \(p. 2016\)](#)
- [Deploy a Model Version from a Different Account \(p. 2017\)](#)
- [View the Deployment History of a Model \(p. 2018\)](#)

## Create a Model Group

A model group contains a group of versioned models. Create a model group by using either the Amazon SDK for Python (Boto3) or in SageMaker Studio.

## Create a Model Package Group (Boto3)

To create a model group by using Boto3, call the `create_model_package_group` method, and specify a name and description as parameters. The following example shows how to create a model group. The response from the `create_model_package_group` call is the Amazon Resource Name (ARN) of the new model package group.

First, import the required packages and set up the SageMaker Boto3 client.

```
import time
import os
from sagemaker import get_execution_role, session
import boto3

region = boto3.Session().region_name

role = get_execution_role()

sm_client = boto3.client('sagemaker', region_name=region)
```

Now create the model group.

```
import time
model_package_group_name = "scikit-iris-detector-" + str(round(time.time()))
model_package_group_input_dict = {
    "ModelPackageName" : model_package_group_name,
    "ModelPackageGroupDescription" : "Sample model package group"
}

create_model_pacakge_group_response =
    sm_client.create_model_package_group(**model_package_group_input_dict)
print('ModelPackageGroup Arn :'
    '{}'.format(create_model_pacakge_group_response['ModelPackageGroupArn']))
```

## Create a Model Package Group (SageMaker Studio)

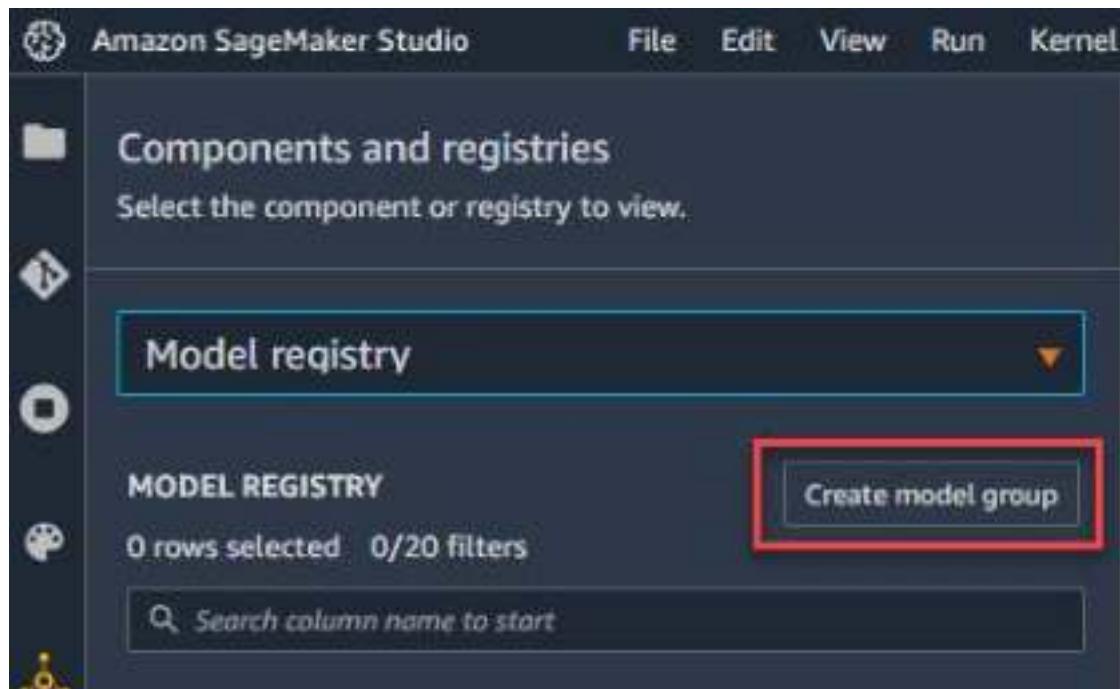
To create a model group in SageMaker Studio, complete the following steps.

1. Sign in to Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
2. In the left navigation pane, choose the **Components and registries** icon ().
3. Choose **Model registry**.

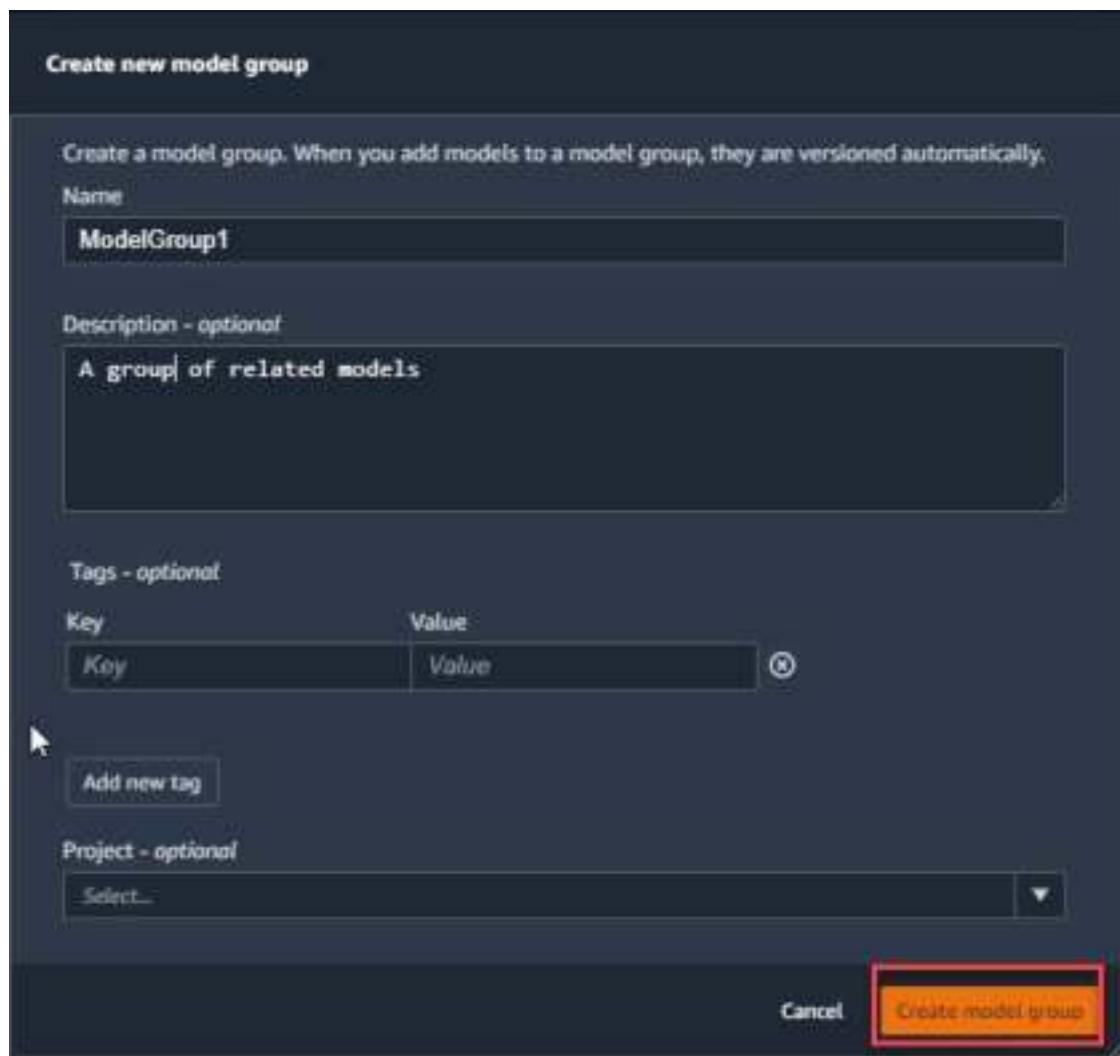
The screenshot shows the Amazon SageMaker Studio interface. At the top, there's a navigation bar with icons for Home, Studio, Projects, Data Wrangler, ML Flow, and Model Registry, followed by 'Amazon SageMaker Studio' and 'File Edit View Run Kernel'. Below this is a sidebar with icons for Home, Studio, Projects, Data Wrangler, ML Flow, and Model Registry. The main area is titled 'Components and registries' with the sub-section 'Select the component or registry to view.' A red box highlights the 'Model registry' button. Below this, there's a 'MODEL REGISTRY' section with a 'Create model group' button. A search bar says 'Search column name to start...'. A table lists three entries: 'ModelGroup1' created '1 hour ago', 'AbaloneMPG-16039329888329896' created '23 days ago', and 'scikit-iris-detector-1016' created '1 month ago'. The table has columns for 'Name' and 'Created'. At the bottom of the list, it says 'End of the list'.

Name	Created
ModelGroup1	1 hour ago
AbaloneMPG-16039329888329896	23 days ago
scikit-iris-detector-1016	1 month ago

4. Choose **Create model group**.



5. In the **Create new model group** dialog box, enter the following information:
  - For **Name**, enter the name of the new model group.
  - (Optional) For **Description**, enter a description for the model group.
  - (Optional) For **Tags**, enter any key-value pairs you want to associate with the model group. For information about using tags, see [Tagging Amazon resources](#) in the *Amazon General Reference*.
  - (Optional) For **Project**, choose a project with which to associate the model group. For information about projects, see [Automate MLOps with SageMaker Projects \(p. 2232\)](#).
6. Choose **Create model group**.



## Register a Model Version

You register a model by creating a model version that specifies the model group to which it belongs. A model version must include both the model artifacts (the trained weights of a model), and the inference code for the model. Create a model version by using either the Amazon SDK for Python (Boto3) or by creating a step in a SageMaker mode building pipeline.

### Register a Model Version (SageMaker Pipelines)

To register a model version by using a SageMaker model building pipeline, create a `RegisterModel` step in your pipeline. For information about creating `RegisterModel` step as part of a pipeline, see [Define a Pipeline \(p. 2201\)](#).

### Register a Model Version (Boto3)

To register a model version by using Boto3, call the `create_model_package` method.

First, you set up the parameter dictionary to pass to the `create_model_package` method.

```
modelpackage_inference_specification = {
    "InferenceSpecification": {
        "Containers": [
            {
                "Image": '257758044811.dkr.ecr.us-east-2.amazonaws.com/sagemaker-xgboost:1.2-1',
            }
        ],
        "SupportedContentTypes": [ "text/csv" ],
        "SupportedResponseMIMETypes": [ "text/csv" ],
    }
}

# Specify the model data
modelpackage_inference_specification["InferenceSpecification"]["Containers"][0]
["ModelDataUrl"] = model_url

create_model_package_input_dict = {
    "ModelPackageGroupName" : model_package_group_name,
    "ModelPackageDescription" : "Model to detect 3 different types of irises (Setosa, Versicolour, and Virginica)",
    "ModelApprovalStatus" : "PendingManualApproval"
}
create_model_package_input_dict.update(modelpackage_inference_specification)
```

Then you call the `create_model_package` method, passing in the parameter dictionary that you just set up.

```
create_mode_package_response =
sm_client.create_model_package(**create_model_package_input_dict)
model_package_arn = create_mode_package_response["ModelPackageArn"]
print('ModelPackage Version ARN : {}'.format(model_package_arn))
```

## View Model Groups and Versions

Model groups and versions help you organize your models. You can view a list of the model versions in a model group.

### View a List of Model Versions in a Group

You can view all of the model versions that are associated with a model group. If a model group represents all models that you train to address a specific ML problem, you can view all of those related models.

#### View a List of Model Versions in a Group (Boto3)

To view model versions associated with a model group by using Boto3, call the `list_model_packages` method, and pass the name of the model group as the value of the `ModelPackageGroupName` parameter. The following code lists the model versions associated with the model group you created in [Create a Model Package Group \(Boto3\) \(p. 2004\)](#).

```
sm_client.list_model_packages(ModelPackageGroupName=model_package_group_name)
```

#### View a List of Model Versions in a Group (SageMaker Studio)

To view a list of the model versions in a model group, complete the following steps.

1. Sign in to Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
2. In the left navigation pane, choose the **Components and registries** icon ( ).
3. Choose **Model registry**.

The screenshot shows the 'Components and registries' page in Amazon SageMaker Studio. The 'Model registry' section is highlighted with a red box. The interface includes a sidebar with icons for datasets, models, and more. The main area has a search bar and a table listing model groups. The table columns are 'Name' and 'Created'. The listed items are:

Name	Created
ModelGroup1	1 hour ago
AbaloneMPG-16039329888329896	23 days ago
scikit-iris-detector-1016	1 month ago

An 'End of the list' message is displayed at the bottom.

4. From the model groups list, choose the model group you want to view.
5. A new tab appears with a list of the model versions in the model group, as shown following.

The screenshot shows the 'ModelGroup1' tab. It has tabs for 'Versions' and 'Settings'. The 'Versions' tab is selected, showing a table of model versions. The table columns are 'Version', 'Stage', 'Status', 'Short description', and 'Modified by'. The data is as follows:

Version	Stage	Status	Short description	Modified by
1	prod	Approved		SageMakerUser
2	None	Pending		

## View the Details of a Model Version

You can view details of a specific model version by using either the Amazon SDK for Python (Boto3) or by using SageMaker Studio.

### View the Details of a Model Version (Boto3)

To view the details of a model version by using Boto3, complete the following steps.

1. Call the `list_model_packages` method to view the model versions in a model group.

```
sm_client.list_model_packages(ModelPackageGroupName="ModelGroup1")
```

The response is a list of model package summaries. You can get the Amazon Resource Name (ARN) of the model versions from this list.

```
{'ModelPackageSummaryList': [ {'ModelPackageGroupName': 'AbaloneMPG-16039329888329896',
  'ModelPackageVersion': 1,
  'ModelPackageArn': 'arn:aws:sagemaker:us-east-2:123456789012:model-package/
ModelGroup1/1',
  'ModelPackageDescription': 'TestMe',
  'CreationTime': datetime.datetime(2020, 10, 29, 1, 27, 46, 46000, tzinfo=tzlocal()),
  'ModelPackageStatus': 'Completed',
  'ModelApprovalStatus': 'Approved'}],
'ResponseMetadata': {'RequestId': '12345678-abcd-1234-abcd-aabbccddeeff',
'HTTPStatusCode': 200,
'HTTPHeaders': {'x-amzn-requestid': '12345678-abcd-1234-abcd-aabbccddeeff',
'content-type': 'application/x-amz-json-1.1',
'content-length': '349',
'date': 'Mon, 23 Nov 2020 04:56:50 GMT'},
'RetryAttempts': 0}}
```

2. Call `describe_model_package` to see the details of the model version. You pass in the ARN of a model version that you got in the output of the call to `list_model_packages`.

```
sm_client.describe_model_package(ModelPackageName="arn:aws:sagemaker:us-
east-2:123456789012:model-package/ModelGroup1/1")
```

The output of this call is a JSON object with the model version details.

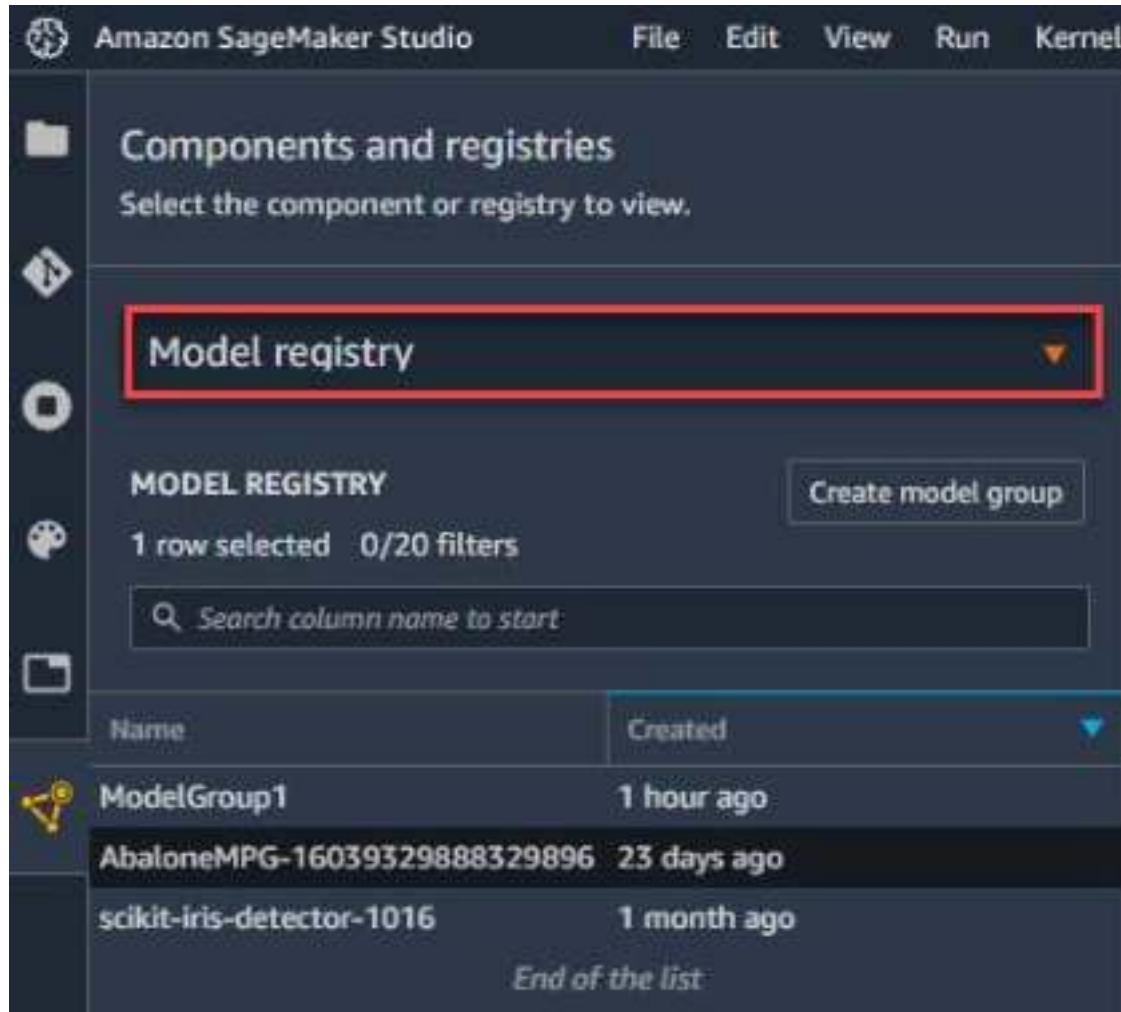
```
{'ModelPackageGroupName': 'ModelGroup1',
'ModelPackageVersion': 1,
'ModelPackageArn': 'arn:aws:sagemaker:us-east-2:123456789012:model-package/
ModelGroup1/1',
'ModelPackageDescription': 'Test Model',
'CreationTime': datetime.datetime(2020, 10, 29, 1, 27, 46, 46000, tzinfo=tzlocal()),
'InferenceSpecification': {'Containers': [{'Image': '257758044811.dkr.ecr.us-
east-2.amazonaws.com/sagemaker-xgboost:1.0-1-cpu-py3',
'ImageDigest':
'sha256:99fa602cff19aee33297a5926f8497ca7bcd2a391b7d600300204eef803bca66',
'ModelDataURL': 's3://sagemaker-us-east-2-123456789012/ModelGroup1/
pipelines-0gdoncek7o9-AbaloneTrain-stmiylhtIR/output/model.tar.gz'}],
'SupportedTransformInstanceTypes': ['ml.m5.xlarge'],
'SupportedRealtimeInferenceInstanceTypes': ['ml.t2.medium', 'ml.m5.xlarge'],
'SupportedContentTypes': ['text/csv'],
'SupportedResponseMIMETypes': ['text/csv']},
'ModelPackageStatus': 'Completed',
'ModelPackageStatusDetails': {'ValidationStatuses': []},
'ImageScanStatuses': []},
```

```
'CertifyForMarketplace': False,  
'ModelApprovalStatus': 'PendingManualApproval',  
'LastModifiedTime': datetime.datetime(2020, 10, 29, 1, 28, 0, 438000,  
tzinfo=tzlocal()),  
'ResponseMetadata': {'RequestId': '12345678-abcd-1234-abcd-aabbccddeeff',  
'HTTPStatusCode': 200,  
'HTTPHeaders': {'x-amzn-requestid': '212345678-abcd-1234-abcd-aabbccddeeff',  
'content-type': 'application/x-amz-json-1.1',  
'content-length': '1038',  
'date': 'Mon, 23 Nov 2020 04:59:38 GMT'},  
'RetryAttempts': 0}}
```

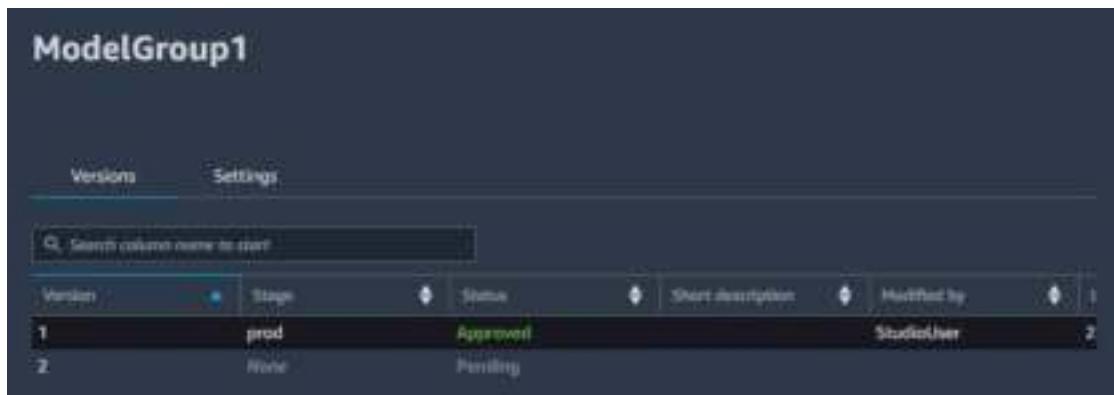
## View the Details of a Model Version (SageMaker Studio)

To view the details of a model version in SageMaker Studio, complete the following steps.

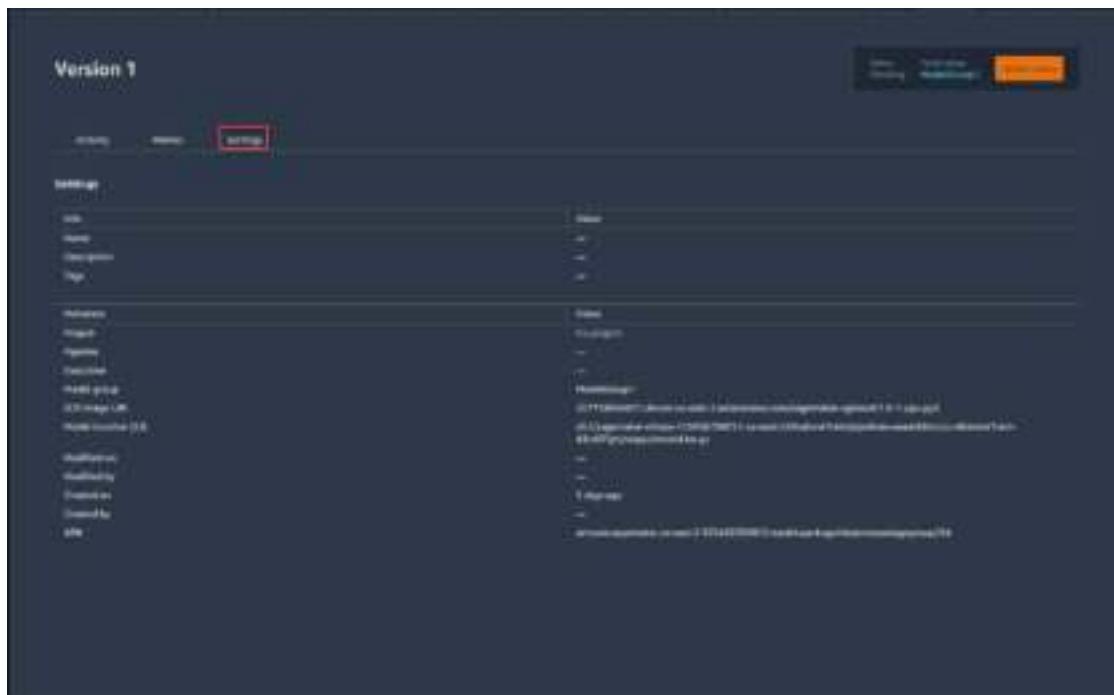
1. Sign in to Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
2. In the left navigation pane, choose the **Components and registries** icon (  ).
3. Choose **Model registry**.



4. From the model groups list, double-click the model group you want to view.
5. A new tab appears with a list of the model versions in the model group.



6. In the list of model versions, double-click the model version for which you want to view details.
  7. On the model version tab that opens, choose one of the following to see details about the model version:
    - **Activity** - Shows events for the model version, such as approval status updates.
    - **Metrics** - Shows quality metrics for the model. For metrics to appear, you must enable data capture for your model by using &SM; Model Monitor. For information about capturing data, see [Capture Data \(p. 1977\)](#).
    - **Settings** - Shows information such as the project that the model version is associated with, the pipeline that generated the model, the model group, and the model's location in Amazon S3.



## Update the Approval Status of a Model

After you create a model version, you typically want to evaluate its performance before you deploy it to a production endpoint. If it performs to your requirements, you can update the approval status of the model version to **Approved**. Setting the status to **Approved** can trigger CI/CD deployment for the model.

model. If the model version does not perform to your requirements, you can update the approval status to **Rejected**.

You can manually update the approval status of a model version after you register it, or you can create a condition step to evaluate the model when you create a SageMaker pipeline. For information about creating a condition step in a SageMaker pipeline, see [Pipeline Steps \(p. 2183\)](#).

When you use one of the SageMaker provided project templates and the approval status of a model version changes, the following action occurs. Only valid transitions are shown.

- **PendingManualApproval** to **Approved** – Triggers CI/CD deployment for the approved model version
- **PendingManualApproval** to **Rejected** – No action
- **Rejected** to **Approved** – Triggers CI/CD deployment for the approved model version
- **Approved** to **Rejected** – Triggers CI/CD to deploy the latest model version with an **Approved** status

You can update the approval status of a model version by using the Amazon SDK for Python (Boto3) or by using SageMaker Studio. You can also update the approval status of a model version as part of a condition step in a SageMaker pipeline. For information about using a model approval step in a SageMaker pipeline, see [SageMaker Pipelines Overview \(p. 2179\)](#).

## Update the Approval Status of a Model (Boto3)

When you created the model version in [Register a Model Version \(p. 2007\)](#), you set the `ModelApprovalStatus` to `PendingManualApproval`. You update the approval status for the model by calling `update_model_package`. Note that you can automate this process by writing code that, for example, sets the approval status of a model depending on the result of an evaluation of some measure of the model's performance. You can also create a step in a pipeline that automatically deploys a new model version when it is approved. The following code snippet shows how to manually change the approval status to `Approved`.

```
model_package_update_input_dict = {  
    "ModelPackageArn" : model_package_arn,  
    "ModelApprovalStatus" : "Approved"  
}  
model_package_update_response =  
    sm_client.update_model_package(**model_package_update_input_dict)
```

## Update the Approval Status of a Model (SageMaker Studio)

The following procedure shows how to manually change the approval status from `Approved` to `Rejected`.

1. Sign in to Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
2. In the left navigation pane, choose the **Components and registries** icon (  ).
3. Choose **Model registry**.

The screenshot shows the 'Components and registries' section of the Amazon SageMaker Studio interface. A red box highlights the 'Model registry' button. Below it, the 'MODEL REGISTRY' section displays a table with three rows:

Name	Created
ModelGroup1	1 hour ago
AbaloneMPG-16039329888329896	23 days ago
scikit-iris-detector-1016	1 month ago

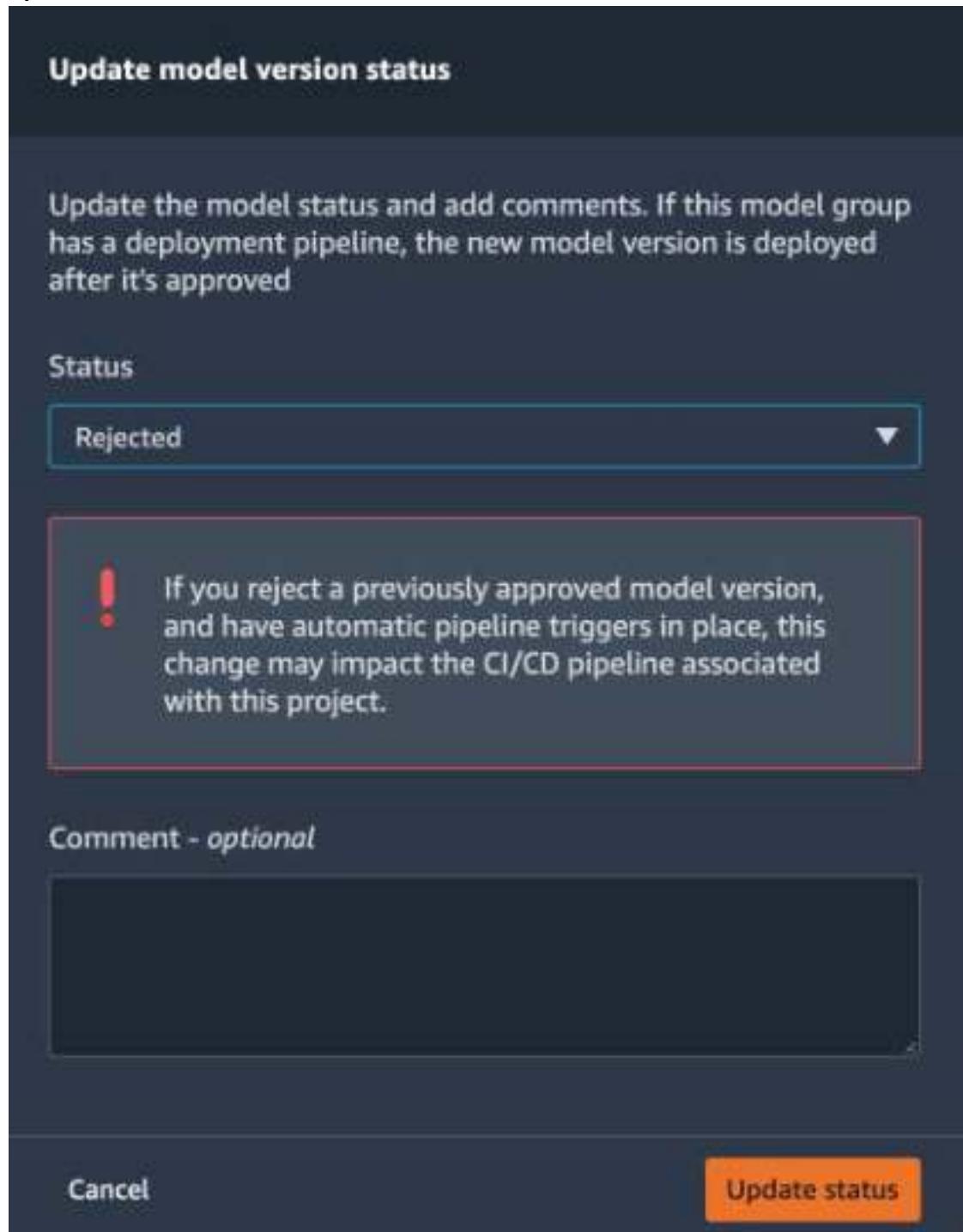
An 'End of the list' message is visible at the bottom right of the table area.

- From the model groups list, choose the model group you want to view. A new tab opens with a list of the model versions in the group.
- In the list of model versions, right-click the model version you want to update and choose **Update model version status**.

A context menu is open over a row in the 'MODEL REGISTRY' table. The 'Status' column for the first row is highlighted with a yellow background. The menu options include:

- Update model version status... (highlighted)
- Open model version
- Copy cell contents
- Shift+Right Click for Browser Menu

6. In the **Update model version status** dialog box, for **Status** choose **Rejected**, and then choose **Update status**.



# Deploy a Model in the Registry

After you register a model version and approve it for deployment, deploy it to a SageMaker endpoint for real-time inference.

When you create an MLOps project and choose an MLOps project template that includes model deployment, approved model versions in the model registry are automatically deployed to production. For information about using SageMaker MLOps projects, see [Automate MLOps with SageMaker Projects \(p. 2232\)](#).

## Deploy a Model in the Registry (Boto3)

To deploy a model version, complete the following steps:

1. Create a model object from the model version by calling `create_model`, passing the Amazon Resource Name (ARN) of the model version as the primary container for the model object. The following code snippet assumes you have already created the SageMaker Boto3 client `sm_client`, and that you have already created a model version with an ARN that you have stored in a variable named `model_version_arn`.

```
model_name = 'DEMO-modelregistry-model-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("Model name : {}".format(model_name))
primary_container = {
    'ModelPackageName': model_version_arn,
}
create_model_response = sm_client.create_model(
    ModelName = model_name,
    ExecutionRoleArn = role,
    PrimaryContainer = primary_container
)
print("Model arn : {}".format(create_model_response["ModelArn"]))
```

2. Create an endpoint configuration by calling `create_endpoint_config`. The endpoint configuration specifies the number and type of Amazon EC2 instances to use for the endpoint.

```
endpoint_config_name = 'DEMO-modelregistry-EndpointConfig-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print(endpoint_config_name)
create_endpoint_config_response = sm_client.create_endpoint_config(
    EndpointConfigName = endpoint_config_name,
    ProductionVariants=[{
        'InstanceType':'ml.m4.xlarge',
        'InitialVariantWeight':1,
        'InitialInstanceCount':1,
        'ModelName':model_name,
        'VariantName':'AllTraffic'}])
```

3. Create the endpoint by calling `create_endpoint`.

```
endpoint_name = 'DEMO-modelregistry-endpoint-' + strftime("%Y-%m-%d-%H-%M-%S", gmtime())
print("EndpointName={}".format(endpoint_name))

create_endpoint_response = sm_client.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name)
print(create_endpoint_response['EndpointArn'])
```

## Deploy a Model Version from a Different Account

Enable an Amazon account to deploy model versions that were created in a different account by adding a cross-account resource policy. For example, one team in your organization might be responsible for training models, and a different team is responsible for deploying and updating models. When you create resource policies, you apply the policy to the resource to which you want to grant access. For more information about cross-account resource policies in Amazon, see [Cross-account policy evaluation logic](#) in the *Amazon Identity and Access Management User Guide*.

To enable cross-account model deployment in SageMaker, you have to provide a cross-account resource policy for the model group that contains the model versions you want to deploy, the Amazon ECR repository where the inference image for the model group resides, and the Amazon S3 bucket where the model versions are stored. The following example creates cross-account policies for all three of these resources, and applies the policies to the resources.

```
import json

# cross account id to grant access to
cross_account_id = "123456789012"

# 1. Create policy for access to the ECR repository
ecr_repository_policy = {
    'Version': '2012-10-17',
    'Statement': [
        {
            'Sid': 'AddPerm',
            'Effect': 'Allow',
            'Principal': {
                'AWS': f'arn:aws:iam::{cross_account_id}:root'
            },
            'Action': ['ecr:*']
        }
    ]
}

# Convert the ECR policy from JSON dict to string
ecr_repository_policy = json.dumps(ecr_repository_policy)

# Set the new ECR policy
ecr = boto3.client('ecr')
response = ecr.set_repository_policy(
    registryId = account,
    repositoryName = 'decision-trees-sample',
    policyText = ecr_repository_policy
)

# 2. Create policy for access to the S3 bucket
bucket_policy = {
    'Version': '2012-10-17',
    'Statement': [
        {
            'Sid': 'AddPerm',
            'Effect': 'Allow',
            'Principal': {
                'AWS': f'arn:aws:iam::{cross_account_id}:root'
            },
            'Action': 's3:*',
            'Resource': f'arn:aws:s3:::{bucket}/*'
        }
    ]
}

# Convert the policy from JSON dict to string
bucket_policy = json.dumps(bucket_policy)

# Set the new policy
s3 = boto3.client('s3')
```

```
respose = s3.put_bucket_policy(
    Bucket = bucket,
    Policy = bucket_policy)

# 3. Create policy for access to the ModelPackageGroup
model_pacakge_group_policy = {
    'Version': '2012-10-17',
    'Statement': [
        {
            'Sid': 'AddPermModelPackageGroup',
            'Effect': 'Allow',
            'Principal': {
                'AWS': f'arn:aws:iam::{cross_account_id}:root'
            },
            'Action': ['sagemaker:DescribeModelPackageGroup'],
            'Resource': f'arn:aws:sagemaker:{region}:{account}:model-package-group/{model_package_group_name}'
        },
        {
            'Sid': 'AddPermModelPackageVersion',
            'Effect': 'Allow',
            'Principal': {
                'AWS': f'arn:aws:iam::{cross_account_id}:root'
            },
            'Action': ["sagemaker:DescribeModelPackage",
                       "sagemaker>ListModelPackages",
                       "sagemaker:UpdateModelPackage",
                       "sagemaker>CreateModel"],
            'Resource': f'arn:aws:sagemaker:{region}:{account}:model-package/{model_package_group_name}/*'
        }
    ]
}

# Convert the policy from JSON dict to string
model_pacakge_group_policy = json.dumps(model_pacakge_group_policy)

# Set the new policy
response = sm_client.put_model_package_group_policy(
    ModelPackageGroupName = model_package_group_name,
    ResourcePolicy = model_pacakge_group_policy)

print('ModelPackageGroupArn :')
print('{}'.format(create_model_pacakge_group_response['ModelPackageGroupArn']))
print("First Versioned ModelPackageArn: " + model_package_arn)
print("Second Versioned ModelPackageArn: " + model_package_arn2)

print("Success! You are all set to proceed for cross account deployment.")
```

The example assumes that you previously defined the following variables:

- `account` - The account of the authenticated caller.
- `bucket` - The S3 bucket where the model versions are stored.
- `sm_client` - A SageMaker Boto3 client.
- `model_package_group_name` - The model group to which you want to grant access.

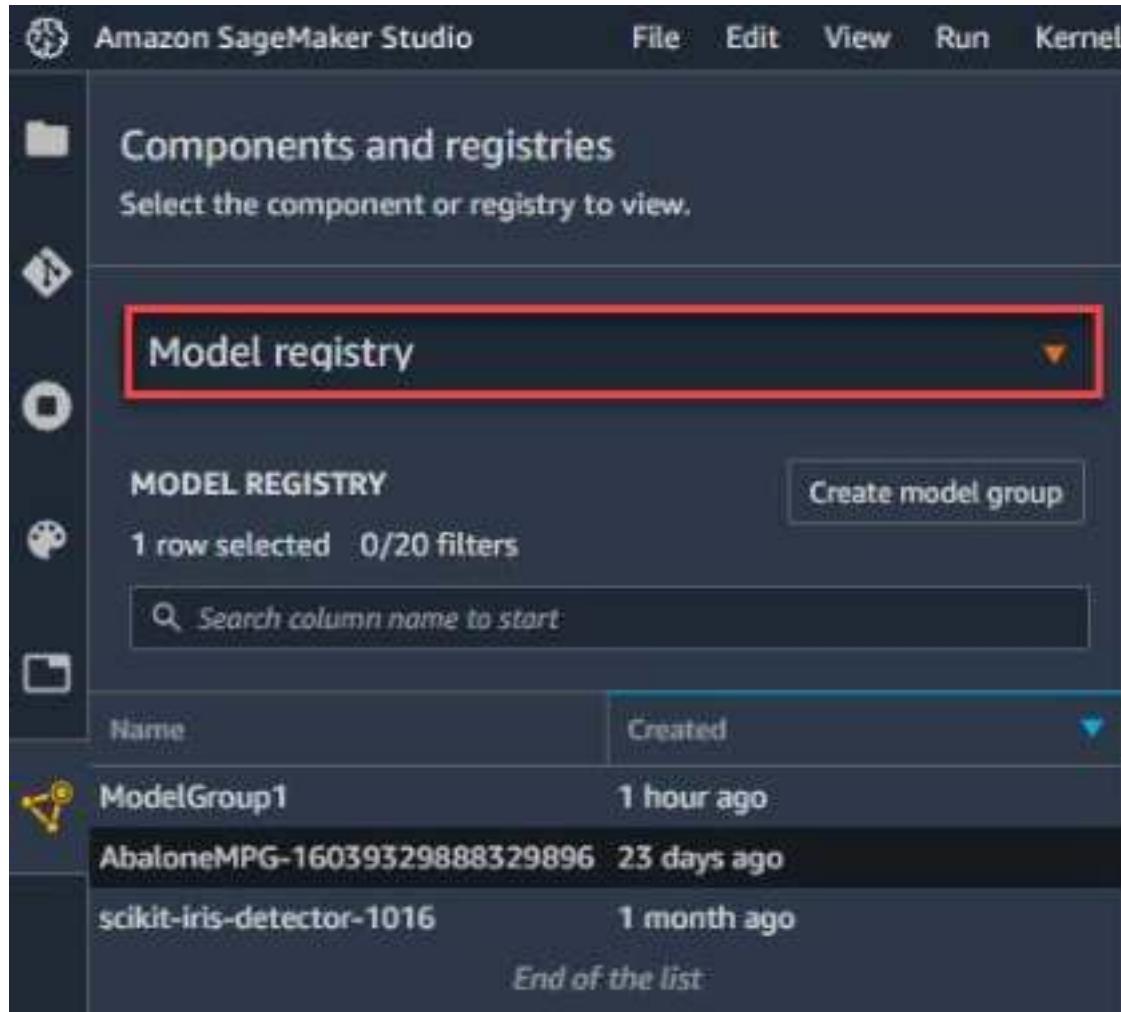
To be able to deploy a model that was created in a different account, the user must have a role that has access to SageMaker actions, such as a role with the `AmazonSageMakerFullAccess` managed policy. For information about SageMaker managed policies, see [Amazon Managed Policies for Amazon SageMaker \(p. 2464\)](#).

## View the Deployment History of a Model

View the deployments for a model version SageMaker Studio by opening the tab for that model version.

### View the deployment history for a model version

1. Sign in to Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
2. In the left navigation pane, choose the **Components and registries** icon (  ).
3. Choose **Model registry**.

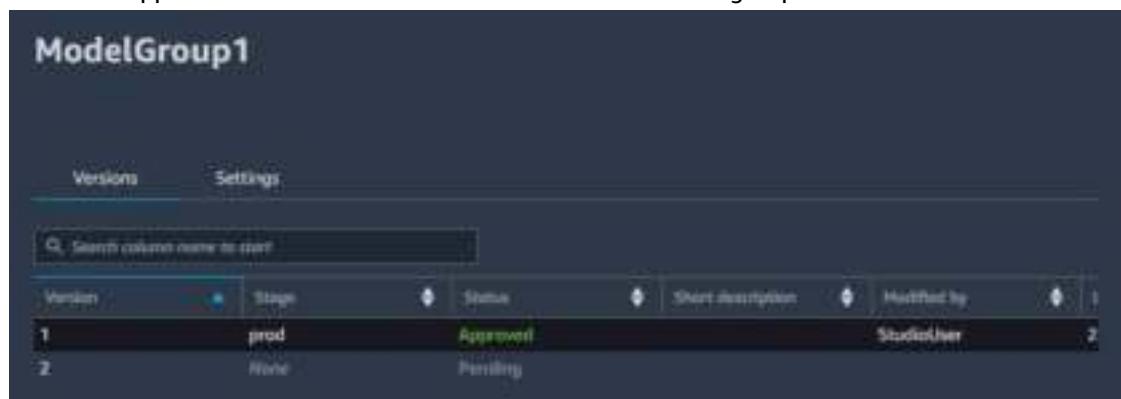


The screenshot shows the 'Components and registries' page in Amazon SageMaker Studio. On the left, there's a sidebar with icons for Model Registry, Model Group, Pipeline, and Data Store. The main area has a title 'Components and registries' and a sub-instruction 'Select the component or registry to view.' Below this, a section titled 'Model registry' is highlighted with a red box. To its right is a 'Create model group' button. The 'MODEL REGISTRY' section displays a table with three rows of data:

Name	Created
ModelGroup1	1 hour ago
AbaloneMPG-16039329888329896	23 days ago
scikit-iris-detector-1016	1 month ago

An 'End of the list' message is at the bottom.

4. From the model groups list, choose the model group you want to view.
5. A new tab appears with a list of the model versions in the model group.



The screenshot shows the 'ModelGroup1' details page. At the top, there are tabs for 'Versions' and 'Settings'. The 'Versions' tab is selected. Below it is a search bar with placeholder text 'Search column name to start...'. The main area is a table titled 'ModelGroup1' with two rows of data:

Version	Stage	Status	Short description	Modified by	
1	prod	Approved		StudioUser	2
2	None	Pending			

6. In the list of model versions, double-click the model version for which you want to view details.

The screenshot shows the 'ModelGroup1' interface. At the top, there are tabs for 'Versions' and 'Settings'. Below the tabs is a search bar with placeholder text 'Search column name or content...'. A table follows, with columns: Version, Stage, Status, Short description, Modified by, and Last modified. The table has two rows:

Version	Stage	Status	Short description	Modified by	Last modified
1	prod	Approved		StudioUser	2023-09-18 10:00:00
2	None	Pending			2023-09-18 10:00:00

7. On the model version tab that opens, choose **Activity**. Deployments for the model version appear as events in the activity list with an **Event type** of **ModelDeployment**.

The screenshot shows the 'Version 1' activity log. At the top, it displays basic information: Status (Approved), Pipeline (MyProject-Neu27-pipeline), Execution (execution-123456789...), Last step (prod), Model group (ModelGroup1), and Version (Version 1). Below this is a navigation bar with tabs: Activity, Metrics, and Settings. The 'Activity' tab is selected. The activity log table has columns: Event type, Event, Component, Modified by, and Last run. It lists several events:

Event type	Event	Component	Modified by	Last run
ModelDeployment	Deployed to stage: prod	prod		20 hours
ModelDeployment	Deployed to stage: staging	staging		22 hours
Approval	Status updated to Approved		unathic-601	22 hours
Approval	Status updated to PendingManualApproval			22 hours

# Compile and Deploy Models with Neo

Neo is a capability of Amazon SageMaker that enables machine learning models to train once and run anywhere in the cloud and at the edge.

If you are a first time user of SageMaker Neo, we recommend you check out the [Getting Started with Edge Devices](#) section to get step-by-step instructions on how to compile and deploy to an edge device.

## What is SageMaker Neo?

Generally, optimizing machine learning models for inference on multiple platforms is difficult because you need to hand-tune models for the specific hardware and software configuration of each platform. If you want to get optimal performance for a given workload, you need to know the hardware architecture, instruction set, memory access patterns, and input data shapes, among other factors. For traditional software development, tools such as compilers and profilers simplify the process. For machine learning, most tools are specific to the framework or to the hardware. This forces you into a manual trial-and-error process that is unreliable and unproductive.

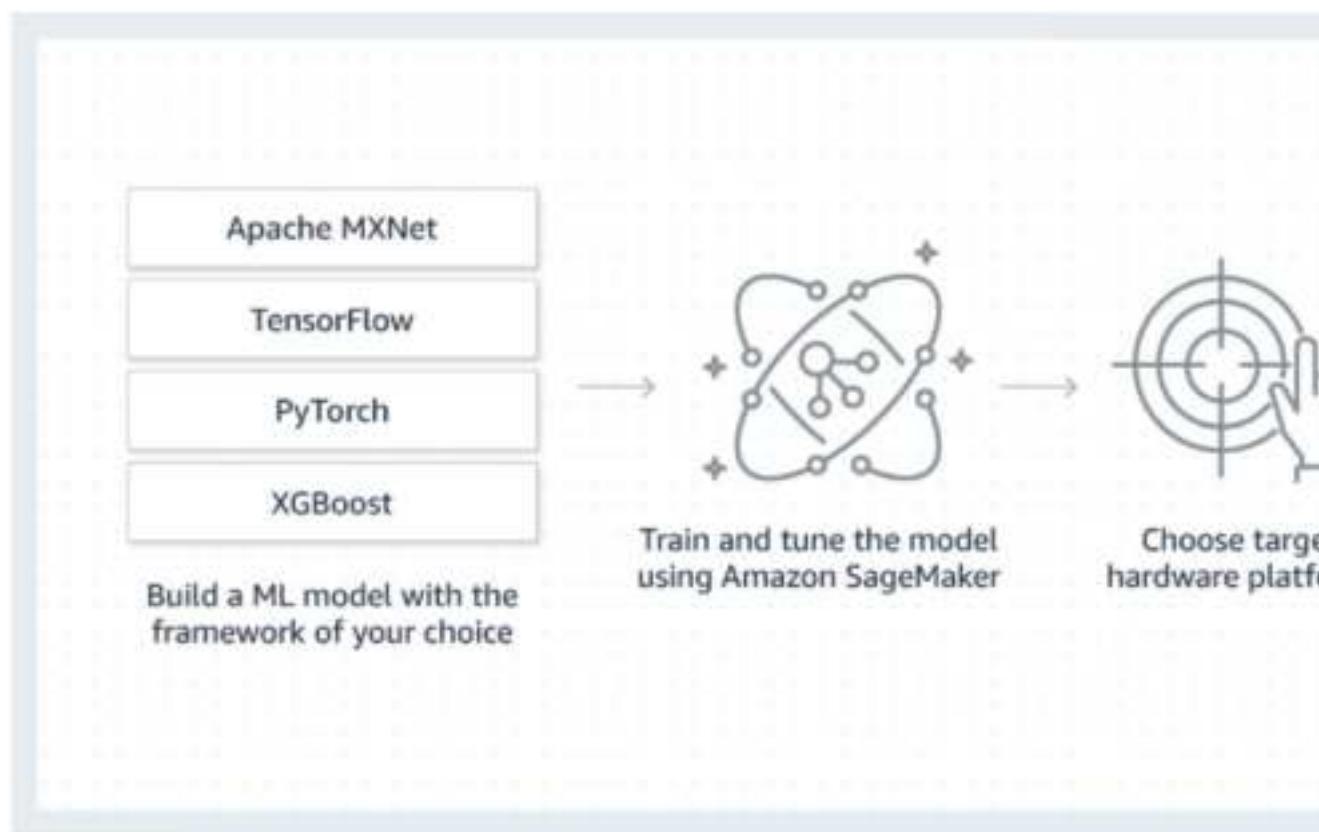
Neo automatically optimizes Gluon, Keras, MXNet, PyTorch, TensorFlow, TensorFlow-Lite, and ONNX models for inference on Android, Linux, and Windows machines based on processors from Ambarella, ARM, Intel, Nvidia, NXP, Qualcomm, Texas Instruments, and Xilinx. Neo is tested with computer vision models available in the model zoos across the frameworks. SageMaker Neo supports compilation and deployment for two main platforms: cloud instances (including Inferentia) and edge devices.

For more information about supported frameworks and cloud instance types you can deploy to, see [Supported Instance Types and Frameworks \(p. 2036\)](#) for cloud instances.

For more information about supported frameworks, edge devices, operating systems, chip architectures, and common machine learning models tested by SageMaker Neo for edge devices, see [Supported Frameworks, Devices, Systems, and Architectures \(p. 2063\)](#) for edge devices.

## How it Works

Neo consists of a compiler and a runtime. First, the Neo compilation API reads models exported from various frameworks. It converts the framework-specific functions and operations into a framework-agnostic intermediate representation. Next, it performs a series of optimizations. Then it generates binary code for the optimized operations, writes them to a shared object library, and saves the model definition and parameters into separate files. Neo also provides a runtime for each target platform that loads and executes the compiled model.



You can create a Neo compilation job from either the SageMaker console, the Amazon Command Line Interface (Amazon CLI), a Python notebook, or the SageMaker SDK. For information on how to compile a model, see [Use Neo to Compile a Model \(p. 2023\)](#). With a few CLI commands, an API invocation, or a few clicks, you can convert a model for your chosen platform. You can deploy the model to a SageMaker endpoint or on an Amazon IoT Greengrass device quickly.

Neo can optimize models with parameters either in FP32 or quantized to INT8 or FP16 bit-width.

## Neo Sample Notebooks

For sample notebooks that use SageMaker Neo to train, compile, optimize, and deploy machine learning models to make inferences, see:

- [MNIST Training, Compilation and Deployment with MXNet Module](#)
- [MNIST Training, Compilation and Deployment with Tensorflow Module](#)
- [Deploying pre-trained PyTorch vision models with SageMaker Neo](#)
- [Model Optimization with an Image Classification Example](#)
- [Model Optimization with XGBoost Example](#)

For instructions on how to run these example notebooks in SageMaker, see [Example Notebooks \(p. 131\)](#). If you need instructions on how to create a notebook instance to run these examples, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#). To navigate to the relevant example in your notebook instance, choose the **Amazon SageMaker Examples** tab to see a list of all of the SageMaker samples. To open a notebook, choose its **Use** tab, then choose **Create copy**.

## Use Neo to Compile a Model

This section shows how to create, describe, stop, and list compilation jobs. The following options are available in Amazon SageMaker Neo for managing the compilation jobs for machine learning models: the Neo CLI, the Amazon SageMaker console, or the Amazon SageMaker SDK.

### Topics

- [Prepare Model for Compilation \(p. 2023\)](#)
- [Compile a Model \(Amazon Command Line Interface\) \(p. 2028\)](#)
- [Compile a Model \(Amazon SageMaker Console\) \(p. 2031\)](#)
- [Compile a Model \(Amazon SageMaker SDK\) \(p. 2035\)](#)

## Prepare Model for Compilation

SageMaker Neo requires machine learning models satisfy specific input data shape. The input shape required for compilation depends on the deep learning framework you use. Once your model input shape is correctly formatted, save your model according to the requirements below. Once you have a saved model, compress the model artifacts.

### Topics

- [What input data shapes does SageMaker Neo expect? \(p. 2023\)](#)
- [Saving Models for SageMaker Neo \(p. 2024\)](#)

## What input data shapes does SageMaker Neo expect?

Before you compile your model, make sure your model is formatted correctly. Neo expects the name and shape of the expected data inputs for your trained model with JSON format or list format. The expected inputs are framework specific.

Below are the input shapes SageMaker Neo expects:

### Keras

Specify the name and shape (NCHW format) of the expected data inputs using a dictionary format for your trained model. Note that while Keras model artifacts should be uploaded in NHWC (channel-last) format, DataInputConfig should be specified in NCHW (channel-first) format. The dictionary formats required are as follows:

- For one input: `{'input_1':[1,3,224,224]}`
- For two inputs: `{'input_1': [1,3,224,224], 'input_2':[1,3,224,224]}`

### MXNet/ONNX

Specify the name and shape (NCHW format) of the expected data inputs using a dictionary format for your trained model. The dictionary formats required are as follows:

- For one input: `{'data':[1,3,1024,1024]}`
- For two inputs: `{'var1': [1,1,28,28], 'var2':[1,1,28,28]}`

## PyTorch

Specify the name and shape (NCHW format) of the expected data inputs using a dictionary format for your trained model. Alternatively, you can specify the shape only using a list format. The dictionary formats required are as follows:

- For one input in dictionary format: `{'input0':[1,3,224,224]}`
- For one input in list format: `[[1,3,224,224]]`
- For two inputs in dictionary format: `{'input0':[1,3,224,224], 'input1':[1,3,224,224]}`
- For two inputs in list format: `[[1,3,224,224], [1,3,224,224]]`

## TensorFlow

Specify the name and shape (NHWC format) of the expected data inputs using a dictionary format for your trained model. The dictionary formats required are as follows:

- For one input: `{'input':[1,1024,1024,3]}`
- For two inputs: `{'data1': [1,28,28,1], 'data2':[1,28,28,1]}`

## TFLite

Specify the name and shape (NHWC format) of the expected data inputs using a dictionary format for your trained model. The dictionary formats required are as follows:

- For one input: `{'input':[1,224,224,3]}`

## XGBoost

An input data name and shape are not needed.

## Saving Models for SageMaker Neo

The following code examples show how to save your model to make it compatible with Neo. Models must be packaged as compressed tar files (\*.tar.gz).

### Keras

Keras models require one model definition file (.h5).

There are two options for saving your Keras model in order to make it compatible for SageMaker Neo:

1. Export to .h5 format with `model.save("<model-name>", save_format="h5")`.
2. Freeze the SavedModel after exporting.

Below is an example of how to export a `tf.keras` model as a frozen graph (option two):

```
import os
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras import backend

tf.keras.backend.set_learning_phase(0)
model = tf.keras.applications.ResNet50(weights='imagenet', include_top=False,
    input_shape=(224, 224, 3), pooling='avg')
model.summary()
```

```
# Save as a SavedModel
export_dir = 'saved_model/'
model.save(export_dir, save_format='tf')

# Freeze saved model
input_node_names = [inp.name.split(":")[0] for inp in model.inputs]
output_node_names = [output.name.split(":")[0] for output in model.outputs]
print("Input names: ", input_node_names)
with tf.Session() as sess:
    loaded = tf.saved_model.load(sess, export_dir=export_dir, tags=["serve"])
    frozen_graph = tf.graph_util.convert_variables_to_constants(sess,
                                                               sess.graph.as_graph_def(),
                                                               output_node_names)
    tf.io.write_graph(graph_or_graph_def=frozen_graph, logdir=".", name="frozen_graph.pb",
                     as_text=False)
```

### Warning

Do not export your model with the `SavedModel` class using `model.save(<path>, save_format='tf')`. This format is suitable for training, but it is not suitable for inference.

## MXNet

MXNet models must be saved as a single symbol file `*-symbol.json` and a single parameter `*.params` files.

### Gluon Models

Define the neural network using the `HybridSequential` Class. This will run the code in the style of symbolic programming (as opposed to imperative programming).

```
from mxnet import nd, sym
from mxnet.gluon import nn

def get_net():
    net = nn.HybridSequential() # Here we use the class HybridSequential.
    net.add(nn.Dense(256, activation='relu'),
           nn.Dense(128, activation='relu'),
           nn.Dense(2))
    net.initialize()
    return net

# Define an input to compute a forward calculation.
x = nd.random.normal(shape=(1, 512))
net = get_net()

# During the forward calculation, the neural network will automatically infer
# the shape of the weight parameters of all the layers based on the shape of
# the input.
net(x)

# hybridize model
net.hybridize()

# export model
net.export('<model_name>') # this will create model-symbol.json and model-0000.params
                           # files

for name in ["<model_name>-0000.params", "<model_name>-symbol.json"]:
    tar.add(name)
tar.close()
```

For more information about hybridizing models, see the [MXNet hybridize documentation](#).

## Gluon Model Zoo (GluonCV)

GluonCV model zoo models come pre-hybridized. So you can just export them.

```
import numpy as np
import mxnet as mx
import gluoncv as gcv

net = gcv.model_zoo.get_model('<model_name>', pretrained=True)
net.export('<model_name>')
tar = tarfile.open("<model_name>.tar.gz", "w:gz")
>>>>> ngl_neo_framework_model_saving_how_to

for name in ["<model_name>-0000.params", "<model_name>-symbol.json"]:
    tar.add(name)
tar.close()
```

## Non Gluon Models

All non-Gluon models when saved to disk use `*-symbol` and `*.params` files. They are therefore already in the correct format for Neo.

```
# Pass the following 3 parameters: sym, args, aux
mx.model.save_checkpoint('<model_name>', 0, sym, args, aux) # this will create
# <model_name>-symbol.json and <model_name>-0000.params files

tar = tarfile.open("<model_name>.tar.gz", "w:gz")

for name in ["<model_name>-0000.params", "<model_name>-symbol.json"]:
    tar.add(name)
tar.close()
```

## PyTorch

PyTorch models must be saved as a definition file (`.pt` or `.pth`) with input datatype of `float32`.

To save your model, use `torch.jit.trace` followed by `torch.save`. This will save an object to a disk file and by default uses python pickle (`pickle_module=pickle`) to save the objects and some metadata. Next, convert the saved model to a compressed tar file.

```
import torchvision
import torch

model = torchvision.models.resnet18(pretrained=True)
model.eval()
inp = torch.rand(1, 3, 224, 224)
model_trace = torch.jit.trace(model, inp)

# Save your model. The following code saves it with the .pth file extension
model_trace.save('model.pth')

# Save as a compressed tar file
with tarfile.open('model.tar.gz', 'w:gz') as f:
    f.add('model.pth')
f.close()
```

## TensorFlow

TensorFlow requires one `.pb` or one `.pbtxt` file and a `variables` directory that contains variables. For frozen models, only one `.pb` or `.pbtxt` file is required.

## Pre-Trained Model

The following code example shows how to use the tar Linux command to compress your model. Run the following in your terminal or in a Jupyter notebook (if you use a Jupyter notebook, insert the ! magic command at the beginning of the statement):

```
import os
import tensorflow as tf

# Download SSD_Mobilenet trained model
wget http://download.tensorflow.org/models/object_detection/
ssd_mobilenet_v2_coco_2018_03_29.tar.gz

# unzip the compressed tar file
tar xvf ssd_mobilenet_v2_coco_2018_03_29.tar.gz

# Change into the directory where the model is stored
cd ssd_mobilenet_v2_coco_2018_03_29

# Compress the tar file and save it in a directory called 'saved_model'
tar czvf panorama-model.tar.gz saved_model
```

The command flags used in this example accomplish the following:

- c: Create an archive
- z: Compress the archive with gzip
- v: Display archive progress
- f: Specify the filename of the archive

## Built-In Estimators

Built-in estimators are either made by framework-specific containers or algorithm-specific containers. Estimator objects for both the built-in algorithm and framework-specific estimator saves the model in the correct format for you when you train the model using the built-in .fit method.

For example, you can use a `sagemaker.TensorFlow` to define a TensorFlow estimator:

```
from sagemaker.tensorflow import TensorFlow

estimator = TensorFlow(entry_point='mnist.py',
                      role=role,
                      framework_version='1.15.3',
                      py_version='py3',
                      training_steps=1000,
                      evaluation_steps=100,
                      instance_count=2,
                      instance_type='ml.c4.xlarge')
```

Then train the model with `.fit` built-in method:

```
estimator.fit(inputs)
```

Before finally compiling model with the build in `compile_model` method:

```
# Specify output path of the compiled model
output_path = '/'.join(estimator.output_path.split('/')[-1:])

# Compile model
optimized_estimator = estimator.compile_model(target_instance_family='ml_c5',
```

```
    input_shape={'data':[1, 784]}, # Batch size 1, 3 channels,  
224x224 Images.  
    output_path=output_path,  
    framework='tensorflow', framework_version='1.15.3')
```

You can also use the `sagemaker.estimator.Estimator` Class to initialize an estimator object for training and compiling a built-in algorithm with the `compile_model` method from the SageMaker Python SDK:

```
from sagemaker.image_uris import retrieve  
aws_region = sagemaker_session.boto_region_name  
  
# Specify built-in algorithm training image  
training_image = retrieve(framework='image-classification',  
                           region=aws_region, image_scope='training')  
  
training_image = retrieve(framework='image-classification', region=aws_region,  
                           image_scope='training')  
  
# Create estimator object for training  
estimator = sagemaker.estimator.Estimator(image_uri=training_image,  
                                           role=role,  
                                           instance_count=1,  
                                           instance_type='ml.p3.8xlarge',  
                                           volume_size = 50,  
                                           max_run = 360000,  
                                           input_mode= 'File',  
                                           output_path=s3_training_output_location,  
                                           base_job_name='image-classification-training'  
)  
  
# Setup the input data_channels to be used later for training.  
  
train_data = sagemaker.inputs.TrainingInput(s3_training_data_location,  
                                             content_type='application/x-recordio',  
                                             s3_data_type='S3Prefix')  
validation_data = sagemaker.inputs.TrainingInput(s3_validation_data_location,  
                                                content_type='application/x-recordio',  
                                                s3_data_type='S3Prefix')  
data_channels = {'train': train_data, 'validation': validation_data}  
  
# Train model  
estimator.fit(inputs=data_channels, logs=True)  
  
# Compile model with Neo  
  
optimized_estimator = estimator.compile_model(target_instance_family='ml_c5',  
                                               input_shape={'data':[1, 3, 224, 224],  
'softmax_label':[1]},  
                                               output_path=s3_compilation_output_location,  
                                               framework='mxnet',  
                                               framework_version='1.7')
```

For more information about compiling models with the SageMaker Python SDK, see [Compile a Model \(Amazon SageMaker SDK\) \(p. 2035\)](#).

## Compile a Model (Amazon Command Line Interface)

This section shows how to manage Amazon SageMaker Neo compilation jobs for machine learning models using Amazon Command Line Interface (CLI). You can create, describe, stop, and list the compilation jobs.

## 1. Create a Compilation Job

With the [CreateCompilationJob](#) API operation, you can specify the data input format, the S3 bucket in which to store your model, the S3 bucket to which to write the compiled model, and the target hardware device or platform.

The following table demonstrates how to configure `CreateCompilationJob` API based on whether your target is a device or a platform.

### Device Example

```
{
    "CompilationJobName": "neo-compilation-job-demo",
    "RoleArn": "arn:aws:iam::your-account:role/service-role/AmazonSageMaker-ExecutionRole-yyyymmddThhmmss",
    "InputConfig": {
        "S3Uri": "s3://<your-bucket>/sagemaker/neo-compilation-job-demo-data/train",
        "DataInputConfig": "{ 'data': [1,3,1024,1024] }",
        "Framework": "MXNET"
    },
    "OutputConfig": {
        "S3OutputLocation": "s3://<your-bucket>/sagemaker/neo-compilation-job-demo-data/compile",
        # A target device specification example for a ml_c5 instance family
        "TargetDevice": "ml_c5"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 300
    }
}
```

You can optionally specify the framework version you used with the `FrameworkVersion` field if you used the PyTorch framework to train your model and your target device is a `ml_*` target.

```
{
    "CompilationJobName": "neo-compilation-job-demo",
    "RoleArn": "arn:aws:iam::your-account:role/service-role/AmazonSageMaker-ExecutionRole-yyyymmddThhmmss",
    "InputConfig": {
        "S3Uri": "s3://<your-bucket>/sagemaker/neo-compilation-job-demo-data/train",
        "DataInputConfig": "{ 'data': [1,3,1024,1024] }",
        "Framework": "PYTORCH",
        # The FrameworkVersion field is only supported when compiling for PyTorch
        # framework and ml_* targets,
        # excluding ml_inf. Supported values are 1.4 or 1.5 or 1.6 . Default is
        1.6
        "FrameworkVersion": "1.6"
    },
    "OutputConfig": {
        "S3OutputLocation": "s3://<your-bucket>/sagemaker/neo-compilation-job-demo-data/compile",
        # A target device specification example for a ml_c5 instance family
        "TargetDevice": "ml_c5",
        # When compiling for ml_* instances using PyTorch framework, use the
        "CompilerOptions" field in
        # OutputConfig to provide the correct data type ("dtype") of the model's
        # input. Default assumed is "float32"
        "CompilerOptions": "{ 'dtype': 'long' }"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 300
    }
}
```

```
    }
}
```

**Note**

This API field is only supported for PyTorch.

Platform Example

```
{
    "CompilationJobName": "neo-test-compilation-job",
    "RoleArn": "arn:aws:iam::<your-account>:role/service-role/AmazonSageMaker-
ExecutionRole-yyyymmddThhmmss",
    "InputConfig": {
        "S3Uri": "s3://<your-bucket>/sagemaker/neo-compilation-job-demo-data/
train",
        "DataInputConfig": "{ 'data': [1,3,1024,1024] }",
        "Framework": "MXNET"
    },
    "OutputConfig": {
        "S3OutputLocation": "s3://<your-bucket>/sagemaker/neo-compilation-job-demo-
data/compile",
        "# A target platform configuration example for a p3.2xlarge instance
        "TargetPlatform": {
            "Os": "LINUX",
            "Arch": "X86_64",
            "Accelerator": "NVIDIA"
        },
        "CompilerOptions": "{ 'cuda-ver': '10.0', 'trt-ver': '6.0.1', 'gpu-code':
'sm_70' }"
    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 300
    }
}
```

**Note**

For the `OutputConfig` API operation, the `TargetDevice` and `TargetPlatform` API operations are mutually exclusive. You have to choose one of the two options.

To find the JSON string examples of `DataInputConfig` depending on frameworks, see [What input data shapes Neo expects](#).

For more information about setting up the configurations, see the [InputConfig](#), [OutputConfig](#), and [TargetPlatform](#) API operations in the SageMaker API reference.

2. After you configure the JSON file, run the following command to create the compilation job:

```
aws sagemaker create-compilation-job \
--cli-input-json file://job.json \
--region us-west-2

# You should get CompilationJobArn
```

3. Describe the compilation job by running the following command:

```
aws sagemaker describe-compilation-job \
--compilation-job-name $JOB_NM \
--region us-west-2
```

4. Stop the compilation job by running the following command:

```
aws sagemaker stop-compilation-job \
--compilation-job-name $JOB_NM \
--region us-west-2

# There is no output for compilation-job operation
```

5. List the compilation job by running the following command:

```
aws sagemaker list-compilation-jobs \
--region us-west-2
```

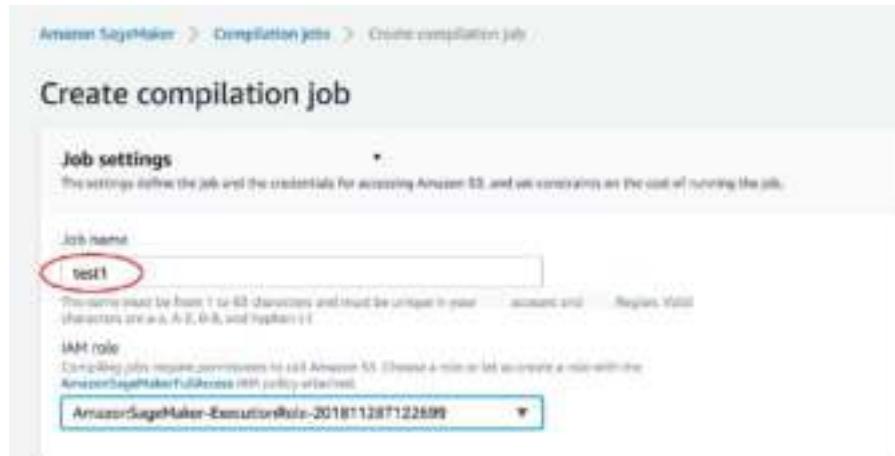
## Compile a Model (Amazon SageMaker Console)

You can create an Amazon SageMaker Neo compilation job in the Amazon SageMaker console.

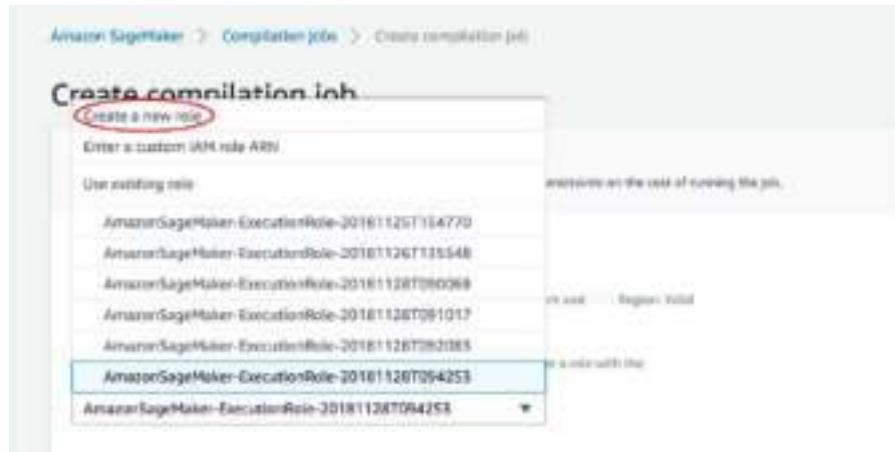
1. In the **Amazon SageMaker** console, choose **Compilation jobs**, and then choose **Create compilation job**.



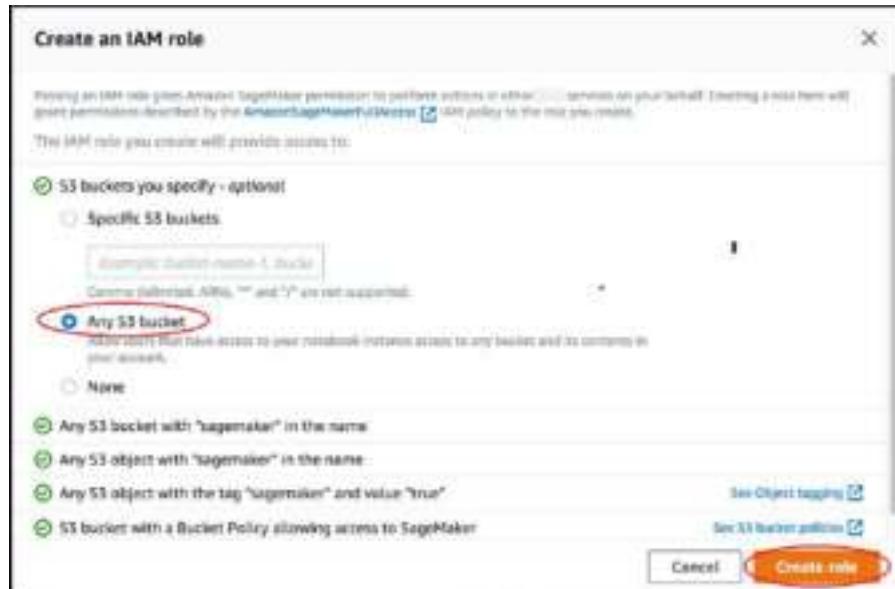
2. On the **Create compilation job** page, under **Job name**, enter a name. Then select an **IAM role**.



3. If you don't have an IAM role, choose **Create a new role**.



- On the **Create an IAM role** page, choose **Any S3 bucket**, and choose **Create role**.



- Non PyTorch Frameworks

Within the **Input configuration** section, enter the full path of the Amazon S3 bucket URI that contains your model artifacts in the **Location of model artifacts** input field. Your model artifacts must be in a compressed tarball file format (.tar.gz).

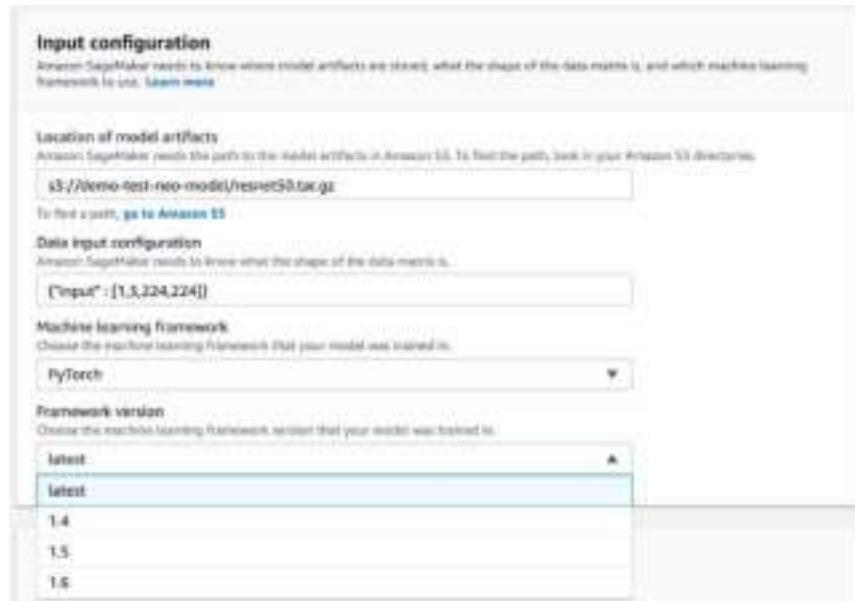
For the **Data input configuration** field, enter the JSON string that specifies the shape of the input data. For **Machine learning framework**, choose the framework of your choice.



To find the JSON string examples of input data shapes depending on frameworks, see [What input data shapes Neo expects](#).

#### PyTorch Framework

Similar instructions apply for compiling PyTorch models. However, if you trained with PyTorch and are trying to compile the model for `m1_*` (except `m1_inf`) target, you can optionally specify the version of PyTorch you used.



To find the JSON string examples of input data shapes depending on frameworks, see [What input data shapes Neo expects](#).

#### Note

When compiling for `m1_*` instances using PyTorch framework, use **Compiler options** field in **Output Configuration** to provide the correct data type (`dtype`) of the model's input. The default is set to "float32".

**Output configuration**  
Amazon SageMaker needs to know where to store the modules compiled with this job. [Learn more](#)

**Target device**  
Choose the target device or the machine learning instance that you want to run your model on after the compilation has completed.

**Target platform**  
Select the target platform that you want your model to run on, such as CPU, architecture, and accelerators.

**Target device**  
Amazon SageMaker needs to know where you intend to deploy your model to an Amazon SageMaker ML instance or to an AWS Greengrass device.

ml\_c5

**Compiler options - optional**  
Specify additional parameters for compiler options in JSON format:  
{"fileType": "TensorRT"}

**S3 Output location**  
Amazon SageMaker needs the path to the S3 bucket or folder where you want to store the compiled module.

s3://bucket-example/detect.tar.gz

To find a path, [go to Amazon S3](#).

**Encryption key - optional**  
Encrypt your data. Choose an existing KMS key or enter a key's ARN:

No Custom Encryption

### Warning

If you specify a Amazon S3 bucket URI path that leads to .pth file, you will receive the following error after starting compilation: **ClientError: InputConfiguration: Unable to untar input model. Please confirm the model is a tar.gz file**

6. Go to the **Output configuration** section. Choose where you want to deploy your model. You can deploy your model to a **Target device** or a **Target platform**. Target devices include cloud and edge devices. Target platforms refer to specific OS, architecture, and accelerators you want your model to run on.

For **S3 Output location**, enter the path to the S3 bucket where you want to store the model. You can optionally add compiler options in JSON format under the **Compiler options** section.

**Output configuration**  
Amazon SageMaker needs to know where to store the modules compiled with this job. [Learn more](#)

**Target device**  
Choose the target device or the machine learning instance that you want to run your model on after the compilation has completed.

**Target platform**  
Select the target platform that you want your model to run on, such as CPU, architecture, and accelerators.

**Target device**  
Amazon SageMaker needs to know where you intend to deploy your model to an Amazon SageMaker ML instance or to an AWS Greengrass device.

Select a target device

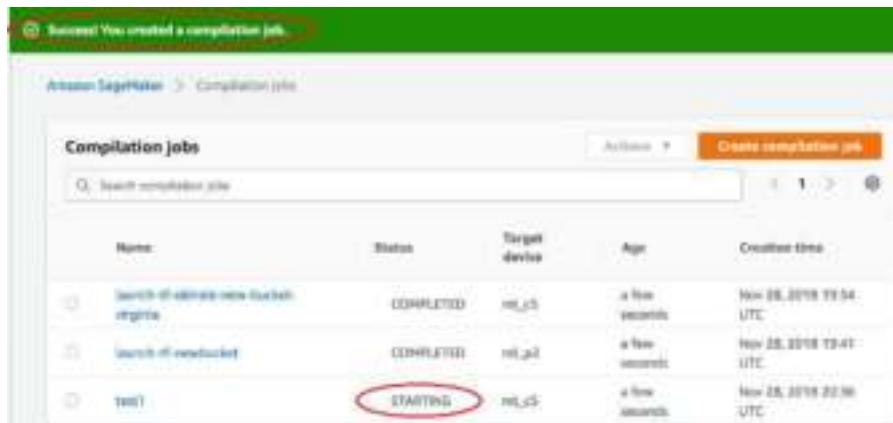
**Compiler options - optional**  
Specify additional parameters for compiler options in JSON format:  
{"fileType": "TensorRT"}

**S3 Output location**  
Amazon SageMaker needs the path to the S3 bucket or folder where you want to store the compiled module.

s3://bucket-example/detect.tar.gz

To find a path, [go to Amazon S3](#).

- Check the status of the compilation job when started. This status of the job can be found at the top of the **Compilation Job** page, as shown in the following screenshot. You can also check the status of it in the **Status** column.



- Check the status of the compilation job when completed. You can check the status in the **Status** column as shown in the following screenshot.



## Compile a Model (Amazon SageMaker SDK)

You can use the `compile_model` API in the [Amazon SageMaker SDK for Python](#) to compile a trained model and optimize it for specific target hardware. The API should be invoked on the estimator object used during model training.

### Note

You must set `MMS_DEFAULT_RESPONSE_TIMEOUT` environment variable to 500 when compiling the model with MXNet or PyTorch. The environment variable is not needed for TensorFlow.

The following is an example of how you can compile a model using the `trained_model_estimator` object:

```
# Replace the value of expected_trained_model_input below and
# specify the name & shape of the expected inputs for your trained model
# in json dictionary form
expected_trained_model_input = {'data':[1, 784]}

# Replace the example target_instance_family below to your preferred target_instance_family
compiled_model = trained_model_estimator.compile_model(target_instance_family='ml_c5',
    input_shape=expected_trained_model_input,
    output_path='insert s3 output path',
    env={'MMS_DEFAULT_RESPONSE_TIMEOUT': '500'})
```

The code compiles the model, saves the optimized model at `output_path`, and creates a SageMaker model that can be deployed to an endpoint. Sample notebooks of using the SDK for Python are provided in the [Neo Model Compilation Sample Notebooks](#) section.

## Cloud Instances

Amazon SageMaker Neo provides compilation support for popular machine learning frameworks such as TensorFlow, PyTorch, MXNet, and more. You can deploy your compiled model to cloud instances and Amazon Inferentia instances. For a full list of supported frameworks and instances types, see [Supported Instances Types and Frameworks](#).

You can compile your model in one of three ways: through the Amazon CLI, the SageMaker Console, or the SageMaker SDK for Python. See, [Use Neo to Compile a Model](#) for more information. Once compiled, your model artifacts are stored in the Amazon S3 bucket URI you specified during the compilation job. You can deploy your compiled model to cloud instances and Amazon Inferentia instances using the SageMaker SDK for Python, Amazon SDK for Python (Boto3), Amazon CLI, or the Amazon console.

If you deploy your model using Amazon CLI, the console, or Boto3, you must select a Docker image Amazon ECR URI for your primary container. See [Neo Inference Container Images](#) for a list of Amazon ECR URIs.

### Topics

- [Supported Instance Types and Frameworks \(p. 2036\)](#)
- [Deploy a Model \(p. 2039\)](#)
- [Request Inferences from a Deployed Service \(p. 2057\)](#)
- [Inference Container Images \(p. 2060\)](#)

## Supported Instance Types and Frameworks

Amazon SageMaker Neo supports popular deep learning frameworks for both compilation and deployment. You can deploy your model to cloud instances, Amazon Inferentia instance types, or Amazon Elastic Inference accelerators.

The following describes frameworks SageMaker Neo supports and the target cloud instances you can compile and deploy to. For information on how to deploy your compiled model to a cloud or Inferentia instance, see [Deploy a Model with Cloud Instances](#). For information on how to deploy your compiled model with Elastic Inference accelerators, see [Use EI on Amazon SageMaker Hosted Endpoints \(p. 1884\)](#).

## Cloud Instances

SageMaker Neo supports the following deep learning frameworks for CPU and GPU cloud instances:

Framework	Framework Version	Model Version	Models	Model Formats (packaged in *.tar.gz)	Toolkits
MXNet	1.8.0	Supports 1.8.0 or earlier	Image Classification, Object Detection, Semantic Segmentation, Pose Estimation,	One symbol file (.json) and one parameter file (.params)	GluonCV v0.8.0

Framework	Framework Version	Model Version	Models	Model Formats (packaged in *.tar.gz)	Toolkits
			Activity Recognition		
ONNX	1.5.0	Supports 1.5.0 or earlier	Image Classification, SVM	One model file (.onnx)	
Keras	2.2.4	Supports 2.2.4 or earlier	Image Classification	One model definition file (.h5)	
PyTorch	1.6	Supports 1.4, 1.5, and 1.6	Image Classification	One model definition file (.pt or .pth) with input dtype of float32	
TensorFlow	1.15.0	Supports 1.15.0 or earlier	Image Classification	For saved models, one .pb or one .pbtxt file and a variables directory that contains variables  For frozen models, only one .pb or .pbtxt file	
TensorFlow-Lite	1.13.1	Supports 1.13.1 or earlier	Image Classification, Object Detection	One model definition flatbuffer file (.tflite)	
XGBoost	0.9	Supports 0.9 or earlier	Decision Trees	One XGBoost model file (.model) where the number of nodes in a tree is less than $2^{31}$	
DARKNET			Image Classification, Object Detection	One config (.cfg) file and one weights (.weights) file	

**Note**

“Model Version” is the version of the framework used to train and export the model.

## Instance Types

You can deploy your SageMaker compiled model to one of the cloud instances listed below:

Instance	Compute Type				
ml_c4	Standard				
ml_c5	Standard				
ml_m4	Standard				
ml_m5	Standard				
ml_p2	Accelerated computing				
ml_p3	Accelerated computing				
ml_g4dn	Accelerated computing				

For information on the available vCPU, memory, and price per hour for each instance type, see [Amazon SageMaker Pricing](#).

### Note

When compiling for ml\_\* instances using PyTorch framework, use **Compiler options** field in **Output Configuration** to provide the correct data type (dtype) of the model's input. The default is set to "float32".

## Amazon Inferentia

SageMaker Neo supports the following deep learning frameworks for Inferentia:

Framework	Framework Version	Model Version	Models	Model Formats (packaged in *.tar.gz)	Toolkits
MXNet	1.5.1	Supports 1.5.1 or earlier	Image Classification, Object Detection, Semantic Segmentation, Pose Estimation, Activity Recognition	One symbol file (.json) and one parameter file (.params)	GluonCV v0.8.0
PyTorch	1.5.1	Supports 1.5.1 or earlier	Image Classification	One model definition file (.pt or .pth) with input dtype of float32	

Framework	Framework Version	Model Version	Models	Model Formats (packaged in *.tar.gz)	Toolkits
TensorFlow	1.15.0	Supports 1.15.0 or earlier	Image Classification	For saved models, one .pb or one .pbtxt file and a variables directory that contains variables  For frozen models, only one .pb or .pbtxt file	

**Note**

"Model Version" is the version of the framework used to train and export the model.

You can deploy your SageMaker Neo-compiled model to Amazon Inferentia-based Amazon EC2 Inf1 instances. Amazon Inferentia is Amazon's first custom silicon chip designed to accelerate deep learning. Currently, you can use the `m1_inf1` instance to deploy your compiled models.

## Amazon Elastic Inference

SageMaker Neo supports the following deep learning frameworks for Elastic Inference:

Framework	Framework Version	Model Version	Models	Model Formats (packaged in *.tar.gz)
TensorFlow	2.3.2	Supports 2.3	Image Classification, Object Detection, Semantic Segmentation, Pose Estimation, Activity Recognition  For saved models, one .pb or one .pbtxt file and a variables directory that contains variables.  For frozen models, only one .pb or .pbtxt file.	

You can deploy your SageMaker Neo-compiled model to an Elastic Inference Accelerator. For more information, see [Use EI on Amazon SageMaker Hosted Endpoints \(p. 1884\)](#).

## Deploy a Model

To deploy an Amazon SageMaker Neo-compiled model to an HTTPS endpoint, you must configure and create the endpoint for the model using Amazon SageMaker hosting services. Currently, developers can use Amazon SageMaker APIs to deploy modules on to `ml.c5`, `ml.c4`, `ml.m5`, `ml.m4`, `ml.p3`, `ml.p2`, and `ml.inf1` instances.

For [Inf1 instances](#), models need to be compiled specifically for ml.inf1 instances. Models compiled for other instance types are not guaranteed to work with ml.inf1 instances.

For [Elastic Inference accelerators](#), models need to be compiled specifically for ml\_eia2 devices. For information on how to deploy your compiled model to an Elastic Inference accelerator, see [Use EI on Amazon SageMaker Hosted Endpoints \(p. 1884\)](#).

When you deploy a compiled model, you need to use the same instance for the target that you used for compilation. This creates a SageMaker endpoint that you can use to perform inferences. You can deploy a Neo-compiled model using any of the following: [Amazon SageMaker SDK for Python](#), [SDK for Python \(Boto3\)](#), [Amazon Command Line Interface](#), and the SageMaker console<https://console.amazonaws.cn/SageMaker>.

#### Note

For deploying a model using Amazon CLI, the console, or Boto3, see [Neo Inference Container Images](#) to select the inference image URI for your primary container.

#### Topics

- [Prerequisites \(p. 2040\)](#)
- [Deploy a Compiled Model Using SageMaker SDK \(p. 2046\)](#)
- [Deploy a Compiled Model Using Boto3 \(p. 2049\)](#)
- [Deploy a Compiled Model Using the Amazon CLI \(p. 2050\)](#)
- [Deploy a Compiled Model Using the Console \(p. 2052\)](#)

## Prerequisites

#### Note

Follow the instructions in this section if you compiled your model using Amazon SDK for Python (Boto3), Amazon CLI, or the SageMaker console.

To create a SageMaker Neo-compiled model, you need the following:

1. A Docker image Amazon ECR URI. You can select one that meets your needs from [this list](#).
2. An entry point script file:

a. **For PyTorch and MXNet models:**

*If you trained your model using SageMaker*, the training script must implement the functions described below. The training script serves as the entry point script during inference. In the example detailed in [MNIST Training, Compilation and Deployment with MXNet Module and SageMaker Neo](#), the training script (`mnist.py`) implements the required functions.

*If you did not train your model using SageMaker*, you need to provide an entry point script (`inference.py`) file that can be used at the time of inference. Based on the framework—MXNet or PyTorch—the inference script location must conform to the SageMaker Python SDK [Model Directory Structure for MxNet](#) or [Model Directory Structure for PyTorch](#).

When using Neo Inference Optimized Container images with **PyTorch** and **MXNet** on CPU and GPU instance types, the inference script must implement the following functions:

- `model_fn`: Loads the model.
- `input_fn`: Converts the incoming request payload into a numpy array.
- `predict_fn`: Performs the prediction.
- `output_fn`: Converts the prediction output into the response payload.
- Alternatively, you can define `transform_fn` to combine `input_fn`, `predict_fn`, and `output_fn`.

The following are examples of `inference.py` script within a directory named `code` (`code/inference.py`) for **PyTorch** and **MXNet (Gluon and Module)**. The examples first load the model and then serve it on image data on a GPU:

#### MXNet Module

```

import numpy as np
import json
import mxnet as mx
import neomx # noqa: F401
from collections import namedtuple

Batch = namedtuple('Batch', ['data'])

# Change the context to mx.cpu() if deploying to a CPU endpoint
ctx = mx.gpu()

def model_fn(model_dir):
    # The compiled model artifacts are saved with the prefix 'compiled'
    sym, arg_params, aux_params = mx.model.load_checkpoint('compiled', 0)
    mod = mx.mod.Module(symbol=sym, context=ctx, label_names=None)
    exe = mod.bind(for_training=False,
                    data_shapes=[('data', (1,3,224,224))],
                    label_shapes=mod._label_shapes)
    mod.set_params(arg_params, aux_params, allow_missing=True)

    # Run warm-up inference on empty data during model load (required for GPU)
    data = mx.nd.empty((1,3,224,224), ctx=ctx)
    mod.forward(Batch([data]))
    return mod

def transform_fn(mod, image, input_content_type, output_content_type):
    # pre-processing
    decoded = mx.image.imdecode(image)
    resized = mx.image.resize_short(decoded, 224)
    cropped, crop_info = mx.image.center_crop(resized, (224, 224))
    normalized = mx.image.color_normalize(cropped.astype(np.float32) / 255,
                                           mean=mx.nd.array([0.485, 0.456, 0.406]),
                                           std=mx.nd.array([0.229, 0.224, 0.225]))
    transposed = normalized.transpose((2, 0, 1))
    batchified = transposed.expand_dims(axis=0)
    casted = batchified.astype(dtype='float32')
    processed_input = casted.as_in_context(ctx)

    # prediction/inference
    mod.forward(Batch([processed_input]))

    # post-processing
    prob = mod.get_outputs()[0].asnumpy().tolist()
    prob_json = json.dumps(prob)
    return prob_json, output_content_type

```

#### MXNet Gluon

```

import numpy as np
import json
import mxnet as mx
import neomx # noqa: F401

# Change the context to mx.cpu() if deploying to a CPU endpoint

```

```

ctx = mx.gpu()

def model_fn(model_dir):
    # The compiled model artifacts are saved with the prefix 'compiled'
    block = mx.gluon.nn.SymbolBlock.imports('compiled-symbol.json',
    ['data'], 'compiled-0000.params', ctx=ctx)

    # Hybridize the model & pass required options for Neo: static_alloc=True &
    static_shape=True
    block.hybridize(static_alloc=True, static_shape=True)

    # Run warm-up inference on empty data during model load (required for GPU)
    data = mx.nd.empty((1,3,224,224), ctx=ctx)
    warm_up = block(data)
    return block


def input_fn(image, input_content_type):
    # pre-processing
    decoded = mx.image.imdecode(image)
    resized = mx.image.resize_short(decoded, 224)
    cropped, crop_info = mx.image.center_crop(resized, (224, 224))
    normalized = mx.image.color_normalize(cropped.astype(np.float32) / 255,
                                           mean=mx.nd.array([0.485, 0.456, 0.406]),
                                           std=mx.nd.array([0.229, 0.224, 0.225]))
    transposed = normalized.transpose((2, 0, 1))
    batchified = transposed.expand_dims(axis=0)
    casted = batchified.astype(dtype='float32')
    processed_input = casted.as_in_context(ctx)
    return processed_input


def predict_fn(processed_input_data, block):
    # prediction/inference
    prediction = block(processed_input_data)
    return prediction


def output_fn(prediction, output_content_type):
    # post-processing
    prob = prediction.numpy().tolist()
    prob_json = json.dumps(prob)
    return prob_json, output_content_type

```

### PyTorch 1.4 and Older

```

import os
import torch
import torch.nn.parallel
import torch.optim
import torch.utils.data
import torch.utils.data.distributed
import torchvision.transforms as transforms
from PIL import Image
import io
import json
import pickle

def model_fn(model_dir):
    """Load the model and return it.
    Providing this function is optional.
    There is a default model_fn available which will load the model
    compiled using SageMaker Neo. You can override it here.

```

```

Keyword arguments:
model_dir -- the directory path where the model artifacts are present
"""

# The compiled model is saved as "compiled.pt"
model_path = os.path.join(model_dir, 'compiled.pt')
with torch.neuro.config(model_dir=model_dir, neo_runtime=True):
    model = torch.jit.load(model_path)
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = model.to(device)

# We recommend that you run warm-up inference during model load
sample_input_path = os.path.join(model_dir, 'sample_input.pkl')
with open(sample_input_path, 'rb') as input_file:
    model_input = pickle.load(input_file)
if torch.is_tensor(model_input):
    model_input = model_input.to(device)
    model(model_input)
elif isinstance(model_input, tuple):
    model_input = (inp.to(device) for inp in model_input if
        torch.is_tensor(inp))
    model(*model_input)
else:
    print("Only supports a torch tensor or a tuple of torch tensors")
    return model


def transform_fn(model, request_body, request_content_type,
                response_content_type):
    """Run prediction and return the output.
    The function
    1. Pre-processes the input request
    2. Runs prediction
    3. Post-processes the prediction output.
    """
    # preprocess
    decoded = Image.open(io.BytesIO(request_body))
    preprocess = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])
    normalized = preprocess(decoded)
    batchified = normalized.unsqueeze(0)
    # predict
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    batchified = batchified.to(device)
    output = model.forward(batchified)

    return json.dumps(output.cpu().numpy().tolist()), response_content_type

```

## PyTorch 1.5 and Newer

Use `neopytorch` to specify the file path of the model if you trained with PyTorch version 1.5 and newer.

```

import os
import torch
import neopytorch
import torch.nn.parallel

```

```

import torch.optim
import torch.utils.data
import torch.utils.data.distributed
import torchvision.transforms as transforms
from PIL import Image
import io
import json
import pickle

def model_fn(model_dir):
    """Load the model and return it.
    Providing this function is optional.
    There is a default model_fn available which will load the model
    compiled using SageMaker Neo. You can override it here.

    Keyword arguments:
    model_dir -- the directory path where the model artifacts are present
    """

    # The compiled model is saved as "compiled.pt"
    model_path = os.path.join(model_dir, 'compiled.pt')
    neopytorch.config(model_dir=model_dir, neo_runtime=True)
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = torch.jit.load(model_path, map_location=device)
    model = model.to(device)

    # We recommend that you run warm-up inference during model load
    sample_input_path = os.path.join(model_dir, 'sample_input.pkl')
    with open(sample_input_path, 'rb') as input_file:
        model_input = pickle.load(input_file)
    if torch.is_tensor(model_input):
        model_input = model_input.to(device)
        model(model_input)
    elif isinstance(model_input, tuple):
        model_input = (inp.to(device)
                      for inp in model_input if torch.is_tensor(inp))
        model(*model_input)
    else:
        print("Only supports a torch tensor or a tuple of torch tensors")
    return model

def transform_fn(model, request_body, request_content_type,
                response_content_type):
    """Run prediction and return the output.
    The function
    1. Pre-processes the input request
    2. Runs prediction
    3. Post-processes the prediction output.
    """
    # preprocess
    decoded = Image.open(io.BytesIO(request_body))
    preprocess = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(
            mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])
    normalized = preprocess(decoded)
    batchified = normalized.unsqueeze(0)

    # predict

```

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
batchified = batchified.to(device)
output = model.forward(batchified)
return json.dumps(output.cpu().numpy().tolist()), response_content_type
```

b. **For inf1 instances or onnx, xgboost, keras container images**

For all other Neo Inference-optimized container images, or inferentia instance types, the entry point script must implement the following functions for Neo Deep Learning Runtime:

- **neo\_preprocess**: Converts the incoming request payload into a numpy array.
- **neo\_postprocess**: Converts the prediction output from Neo Deep Learning Runtime into the response body.

**Note**

The preceding two functions do not use any of the functionalities of MXNet, PyTorch, or TensorFlow.

For examples of how to use these functions, see [Neo Model Compilation Sample Notebooks](#).

c. **For TensorFlow models**

If your model requires custom pre- and post-processing logic before data is sent to the model, then you must specify an entry point script `inference.py` file that can be used at the time of inference. The script should implement either a pair of `input_handler` and `output_handler` functions or a single handler function.

**Note**

Note that if handler function is implemented, `input_handler` and `output_handler` are ignored.

The following is a code example of `inference.py` script that you can put together with the compile model to perform custom pre- and post-processing on an image classification model. The SageMaker client sends the image file as an `application/x-image` content type to the `input_handler` function, where it is converted to JSON. The converted image file is then sent to the [Tensorflow Model Server \(TFX\)](#) using the REST API.

```
import json
import numpy as np
import json
import io
from PIL import Image

def input_handler(data, context):
    """ Pre-process request input before it is sent to TensorFlow Serving REST API

    Args:
        data (obj): the request data, in format of dict or string
        context (Context): an object containing request and configuration details

    Returns:
        (dict): a JSON-serializable dict that contains request body and headers
    """
    f = data.read()
    f = io.BytesIO(f)
    image = Image.open(f).convert('RGB')
    batch_size = 1
    image = np.asarray(image.resize((512, 512)))
    image = np.concatenate([image[np.newaxis, :, :] * batch_size]
```

```

body = json.dumps({"signature_name": "serving_default", "instances":
    image.tolist()})
return body

def output_handler(data, context):
    """Post-process TensorFlow Serving output before it is returned to the client.

    Args:
        data (obj): the TensorFlow serving response
        context (Context): an object containing request and configuration details

    Returns:
        (bytes, string): data to return to client, response content type
    """
    if data.status_code != 200:
        raise ValueError(data.content.decode('utf-8'))

    response_content_type = context.accept_header
    prediction = data.content
    return prediction, response_content_type

```

If there is no custom pre- or post-processing, the SageMaker client converts the file `image` to JSON in a similar way before sending it over to the SageMaker endpoint.

For more information, see the [Deploying to TensorFlow Serving Endpoints in the SageMaker Python SDK](#).

3. The Amazon S3 bucket URI that contains the compiled model artifacts.

## Deploy a Compiled Model Using SageMaker SDK

You must satisfy the [prerequisites](#) section if the model was compiled using Amazon SDK for Python (Boto3), Amazon CLI, or the Amazon SageMaker console. Follow one of the following use cases to deploy a model compiled with SageMaker Neo based on how you compiled your model.

### Topics

- [If you compiled your model using the SageMaker SDK \(p. 2046\)](#)
- [If you compiled your model using MXNet or PyTorch \(p. 2046\)](#)
- [If you compiled your model using Boto3, SageMaker console, or the CLI for TensorFlow \(p. 2048\)](#)

### If you compiled your model using the SageMaker SDK

The `sagemaker.Model` object handle for the compiled model supplies the `deploy()` function, which enables you to create an endpoint to serve inference requests. The function lets you set the number and type of instances that are used for the endpoint. You must choose an instance for which you have compiled your model. For example, in the job compiled in [Compile a Model \(Amazon SageMaker SDK\)](#) section, this is `m1_c5`.

```

predictor = compiled_model.deploy(initial_instance_count = 1, instance_type =
    'ml.c5.4xlarge')

# Print the name of newly created endpoint
print(predictor.endpoint_name)

```

### If you compiled your model using MXNet or PyTorch

Create the SageMaker model and deploy it using the `deploy()` API under the framework-specific Model APIs. For MXNet, it is [MXNetModel](#) and for PyTorch, it is [PyTorchModel](#). When you are creating and deploying an SageMaker model, you must set `MMS_DEFAULT_RESPONSE_TIMEOUT` environment variable

to 500 and specify the `entry_point` parameter as the inference script (`inference.py`) and the `source_dir` parameter as the directory location (code) of the inference script. To prepare the inference script (`inference.py`) follow the Prerequisites step.

The following example shows how to use these functions to deploy a compiled model using the SageMaker SDK for Python:

#### MXNet

```
from sagemaker.mxnet import MXNetModel

# Create SageMaker model and deploy an endpoint
sm_mxnet_compiled_model = MXNetModel(
    model_data='insert S3 path of compiled MXNet model archive',
    role='AmazonSageMaker-ExecutionRole',
    entry_point='inference.py',
    source_dir='code',
    framework_version='1.8.0',
    py_version='py3',
    image_uri='insert appropriate ECR Image URI for MXNet',
    env={'MMS_DEFAULT_RESPONSE_TIMEOUT': '500'},
)

# Replace the example instance_type below to your preferred instance_type
predictor = sm_mxnet_compiled_model.deploy(initial_instance_count = 1, instance_type =
'ml.p3.2xlarge')

# Print the name of newly created endpoint
print(predictor.endpoint_name)
```

#### PyTorch 1.4 and Older

```
from sagemaker.pytorch import PyTorchModel

# Create SageMaker model and deploy an endpoint
sm_pytorch_compiled_model = PyTorchModel(
    model_data='insert S3 path of compiled PyTorch model archive',
    role='AmazonSageMaker-ExecutionRole',
    entry_point='inference.py',
    source_dir='code',
    framework_version='1.4.0',
    py_version='py3',
    image_uri='insert appropriate ECR Image URI for PyTorch',
    env={'MMS_DEFAULT_RESPONSE_TIMEOUT': '500'},
)

# Replace the example instance_type below to your preferred instance_type
predictor = sm_pytorch_compiled_model.deploy(initial_instance_count = 1, instance_type =
'ml.p3.2xlarge')

# Print the name of newly created endpoint
print(predictor.endpoint_name)
```

#### PyTorch 1.5 and Newer

```
from sagemaker.pytorch import PyTorchModel

# Create SageMaker model and deploy an endpoint
sm_pytorch_compiled_model = PyTorchModel(
    model_data='insert S3 path of compiled PyTorch model archive',
    role='AmazonSageMaker-ExecutionRole',
    entry_point='inference.py',
    source_dir='code',
```

```

        framework_version='1.5',
        py_version='py3',
        image_uri='insert appropriate ECR Image URI for PyTorch',
    )

# Replace the example instance_type below to your preferred instance_type
predictor = sm_pytorch_compiled_model.deploy(initial_instance_count = 1, instance_type
= 'ml.p3.2xlarge')

# Print the name of newly created endpoint
print(predictor.endpoint_name)

```

**Note**

The `AmazonSageMakerFullAccess` and `AmazonS3ReadOnlyAccess` policies must be attached to the `AmazonSageMaker-ExecutionRole` IAM role.

**If you compiled your model using Boto3, SageMaker console, or the CLI for TensorFlow**

Construct a `TensorFlowModel` object, then call `deploy`:

```

role='AmazonSageMaker-ExecutionRole'
model_path='S3 path for model file'
framework_image='inference container arn'
tf_model = TensorFlowModel(model_data=model_path,
                           framework_version='1.15.3',
                           role=role,
                           image_uri=framework_image)
instance_type='ml.c5.xlarge'
predictor = tf_model.deploy(instance_type=instance_type,
                           initial_instance_count=1)

```

See [Deploying directly from model artifacts](#) for more information.

You can select a Docker image Amazon ECR URI that meets your needs from [this list](#).

For more information on how to construct a `TensorFlowModel` object, see the [SageMaker SDK](#).

**Note**

Your first inference request might have high latency if you deploy your model on a GPU. This is because an optimized compute kernel is made on the first inference request. We recommend that you make a warm-up file of inference requests and store that alongside your model file before sending it off to a TFX. This is known as "warming up" the model.

The following code snippet demonstrates how to produce the warm-up file for image classification example in the [prerequisites](#) section:

```

import tensorflow as tf
from tensorflow_serving.apis import classification_pb2
from tensorflow_serving.apis import inference_pb2
from tensorflow_serving.apis import model_pb2
from tensorflow_serving.apis import predict_pb2
from tensorflow_serving.apis import prediction_log_pb2
from tensorflow_serving.apis import regression_pb2
import numpy as np

with tf.python_io.TFRecordWriter("tf_serving_warmup_requests") as writer:
    img = np.random.uniform(0, 1, size=[224, 224, 3]).astype(np.float32)
    img = np.expand_dims(img, axis=0)
    test_data = np.repeat(img, 1, axis=0)
    request = predict_pb2.PredictRequest()
    request.model_spec.name = 'compiled_models'

```

```
request.model_spec.signature_name = 'serving_default'
request.inputs['Placeholder:0'].CopyFrom(tf.compat.v1.make_tensor_proto(test_data,
shape=test_data.shape, dtype=tf.float32))
log = prediction_log_pb2.PredictionLog(
predict_log=prediction_log_pb2.PredictLog(request=request))
writer.write(log.SerializeToString())
```

For more information on how to “warm up” your model, see the [TensorFlow TFX page](#).

## Deploy a Compiled Model Using Boto3

You must satisfy the [prerequisites](#) section if the model was compiled using Amazon SDK for Python (Boto3), Amazon CLI, or the Amazon SageMaker console. Follow the steps below to create and deploy a SageMaker Neo-compiled model using [Amazon Web Services SDK for Python \(Boto3\)](#).

### Topics

- [Deploy the Model \(p. 2049\)](#)

### Deploy the Model

After you have satisfied the [prerequisites](#), use the `create_model`, `create_endpoint_config`, and `create_endpoint` APIs.

The following example shows how to use these APIs to deploy a model compiled with Neo:

```
import boto3
client = boto3.client('sagemaker')

# create sagemaker model
create_model_api_response = client.create_model(
    ModelName='my-sagemaker-model',
    PrimaryContainer={
        'Image': <insert the ECR Image URI>,
        'ModelDataUrl': 's3://path/to/model/artifact/
model.tar.gz',
        'Environment': {}
    },
    ExecutionRoleArn='ARN for AmazonSageMaker-
ExecutionRole'
)

print ("create_model API response", create_model_api_response)

# create sagemaker endpoint config
create_endpoint_config_api_response = client.create_endpoint_config(
    EndpointConfigName='sagemaker-neomxnet-
endpoint-configuration',
    ProductionVariants=[
        {
            'VariantName': <provide your variant
name>,
            'ModelName': 'my-sagemaker-model',
            'InitialInstanceCount': 1,
            'InstanceType': <provide your instance
type here>
        },
    ]
)

print ("create_endpoint_config API response", create_endpoint_config_api_response)

# create sagemaker endpoint
```

```
create_endpoint_api_response = client.create_endpoint(
    EndpointName='provide your endpoint name',
    EndpointConfigName=<insert your endpoint config name>,
)
print ("create_endpoint API response", create_endpoint_api_response)
```

### Note

The `AmazonSageMakerFullAccess` and `AmazonS3ReadOnlyAccess` policies must be attached to the `AmazonSageMaker-ExecutionRole` IAM role.

For full syntax of `create_model`, `create_endpoint_config`, and `create_endpoint` APIs, see [create\\_model](#), [create\\_endpoint\\_config](#), and [create\\_endpoint](#), respectively.

If you did not train your model using SageMaker, specify the following environment variables:

MXNet and PyTorch

```
"Environment": {
    "SAGEMAKER_PROGRAM": "inference.py",
    "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code",
    "SAGEMAKER_CONTAINER_LOG_LEVEL": "20",
    "SAGEMAKER_REGION": "insert your region",
    "MMS_DEFAULT_RESPONSE_TIMEOUT": "500"
}
```

TensorFlow

```
"Environment": {
    "SAGEMAKER_PROGRAM": "inference.py",
    "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code",
    "SAGEMAKER_CONTAINER_LOG_LEVEL": "20",
    "SAGEMAKER_REGION": "insert your region"
}
```

If you trained your model using SageMaker, specify the environment variable `SAGEMAKER_SUBMIT_DIRECTORY` as the full Amazon S3 bucket URI that contains the training script.

## Deploy a Compiled Model Using the Amazon CLI

You must satisfy the [prerequisites](#) section if the model was compiled using Amazon SDK for Python (Boto3), Amazon CLI, or the Amazon SageMaker console. Follow the steps below to create and deploy a SageMaker Neo-compiled model using the [Amazon CLI](#).

### Topics

- [Deploy the Model \(p. 2050\)](#)

### Deploy the Model

After you have satisfied the [prerequisites](#), use the `create-model`, `create-endpoint-config`, and `create-endpoint` Amazon CLI commands. The following steps explain how to use these commands to deploy a model compiled with Neo:

### Create a Model

From [Neo Inference Container Images](#), select the inference image URI and then use `create-model` API to create a SageMaker model. You can do this with two steps:

1. Create a `create_model.json` file. Within the file, specify the name of the model, the image URI, the path to the `model.tar.gz` file in your Amazon S3 bucket, and your SageMaker execution role:

```
{
  "ModelName": "insert model name",
  "PrimaryContainer": {
    "Image": "insert the ECR Image URI",
    "ModelDataURL": "insert S3 archive URL",
    "Environment": {"See details below"}
  },
  "ExecutionRoleArn": "ARN for AmazonSageMaker-ExecutionRole"
}
```

If you trained your model using SageMaker, specify the following environment variable:

```
"Environment": {
  "SAGEMAKER_SUBMIT_DIRECTORY" : "[Full S3 path for *.tar.gz file containing the training script]"
}
```

If you did not train your model using SageMaker, specify the following environment variables:  
MXNet and PyTorch

```
"Environment": {
  "SAGEMAKER_PROGRAM": "inference.py",
  "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code",
  "SAGEMAKER_CONTAINER_LOG_LEVEL": "20",
  "SAGEMAKER_REGION": "insert your region",
  "MMS_DEFAULT_RESPONSE_TIMEOUT": "500"
}
```

TensorFlow

```
"Environment": {
  "SAGEMAKER_PROGRAM": "inference.py",
  "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code",
  "SAGEMAKER_CONTAINER_LOG_LEVEL": "20",
  "SAGEMAKER_REGION": "insert your region"
}
```

#### Note

The `AmazonSageMakerFullAccess` and `AmazonS3ReadOnlyAccess` policies must be attached to the `AmazonSageMaker-ExecutionRole` IAM role.

2. Run the following command:

```
aws sagemaker create-model --cli-input-json file://create_model.json
```

For the full syntax of the `create-model` API, see [create-model](#).

## Create an Endpoint Configuration

After creating a SageMaker model, create the endpoint configuration using the `create-endpoint-config` API. To do this, create a JSON file with your endpoint configuration specifications. For example, you can use the following code template and save it as `create_config.json`:

```
{
```

```
"EndpointConfigName": "<provide your endpoint config name>",
"ProductionVariants": [
    {
        "VariantName": "<provide your variant name>",
        "ModelName": "my-sagemaker-model",
        "InitialInstanceCount": 1,
        "InstanceType": "<provide your instance type here>",
        "InitialVariantWeight": 1.0
    }
]
```

Now run the following Amazon CLI command to create your endpoint configuration:

```
aws sagemaker create-endpoint-config --cli-input-json file://create_config.json
```

For the full syntax of the `create-endpoint-config` API, see [create-endpoint-config](#).

### Create an Endpoint

After you have created your endpoint configuration, create an endpoint using the `create-endpoint` API:

```
aws sagemaker create-endpoint --endpoint-name '<provide your endpoint name>' --endpoint-
config-name '<insert your endpoint config name>'
```

For the full syntax of the `create-endpoint` API, see [create-endpoint](#).

### Deploy a Compiled Model Using the Console

You must satisfy the [prerequisites](#) section if the model was compiled using Amazon SDK for Python (Boto3), the Amazon CLI, or the Amazon SageMaker console. Follow the steps below to create and deploy a SageMaker Neo-compiled model using the SageMaker console<https://console.amazonaws.cn/SageMaker>.

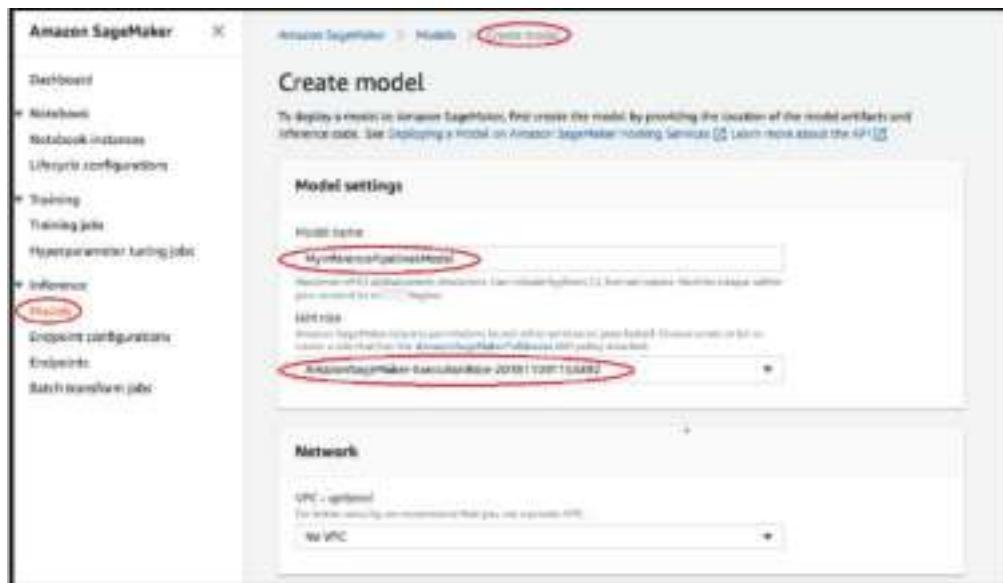
#### Topics

- [Deploy the Model \(p. 2052\)](#)

### Deploy the Model

After you have satisfied the [prerequisites](#), use the following steps to deploy a model compiled with Neo:

1. Choose **Models**, and then choose **Create models** from the **Inference** group. On the **Create model** page, complete the **Model name**, **IAM role**, and **VPC** fields (optional), if needed.



2. To add information about the container used to deploy your model, choose **Add container container**, then choose **Next**. Complete the **Container input options**, **Location of inference code image**, and **Location of model artifacts**, and optionally, **Container host name** and **Environmental variables** fields.

Key	Value	<b>Remove</b>
key1	value1	<b>Remove</b>
key2	value2	<b>Remove</b>

[Add environment variable](#)

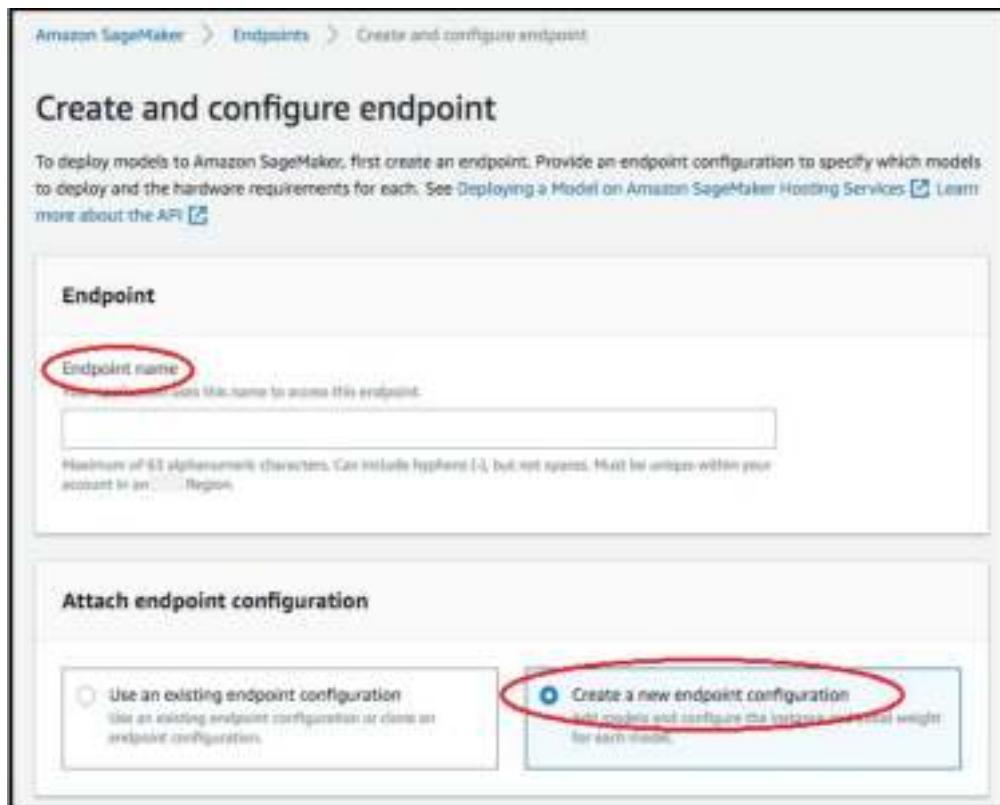
3. To deploy Neo-compiled models, choose the following:
  - **Container input options:** Choose **Provide model artifacts and inference image**.
  - **Location of inference code image:** Choose the inference image URI from [Neo Inference Container Images](#), depending on the Amazon Region and kind of application.
  - **Location of model artifact:** Enter the Amazon S3 bucket URI of the compiled model artifact generated by the Neo compilation API.
  - **Environment variables:**
    - Leave this field blank for **SageMaker XGBoost**.
    - If you trained your model using SageMaker, specify the environment variable `SAGEMAKER_SUBMIT_DIRECTORY` as the Amazon S3 bucket URI that contains the training script.
    - If you did not train your model using SageMaker, specify the following environment variables:

Key	Values for MXNet and PyTorch	Values TensorFlow
<code>SAGEMAKER_PROGRAM</code>	<code>inference.py</code>	<code>inference.py</code>
<code>SAGEMAKER_SUBMIT_DIRECTORY</code>	<code>/opt/ml/model/code</code>	<code>/opt/ml/model/code</code>
<code>SAGEMAKER_CONTAINER_LOG_LEVEL</code>	20	20
<code>SAGEMAKER_REGION</code>	<your region>	<your region>
<code>MMS_DEFAULT_RESPONSE_TIMEOUT</code>		Leave this field blank for TF

4. Confirm that the information for the containers is accurate, and then choose **Create model**. On the [Create model landing page](#), choose **Create endpoint**.



5. In [Create and configure endpoint](#) diagram, specify the **Endpoint name**. For **Attach endpoint configuration**, choose **Create a new endpoint configuration**.



6. In **New endpoint configuration** page, specify the **Endpoint configuration name**.

New endpoint configuration

To deploy models to Amazon SageMaker, first create an endpoint configuration. In the configuration, specify which models to deploy, and the relative traffic weighting and hardware requirements for each.

Endpoint configuration name

Maximum of 63 alphanumeric characters. Can include hyphens (-), but not spaces. Must be unique within your account in all Regions.

Encryption key - optional

Encrypt your data. Choose an existing AWS key or enter a key's ARN.

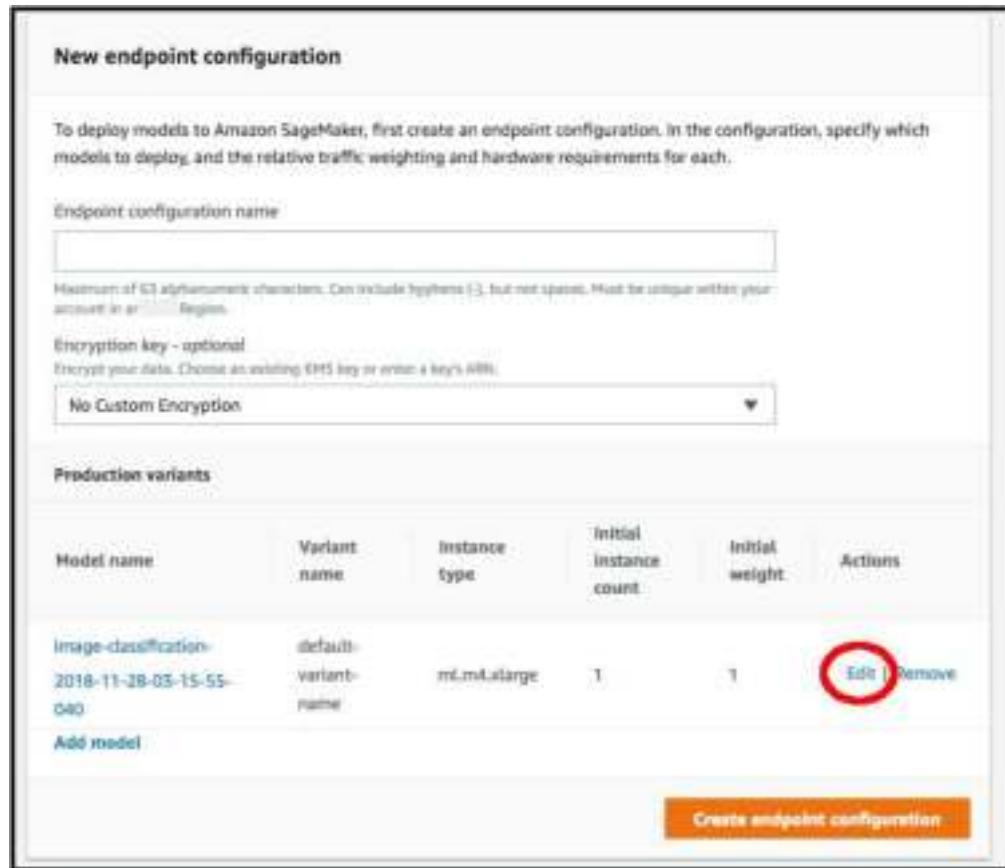
No Custom Encryption

Production variants

Model name	Variant name	Instance type	Initial instance count	Initial weight	Actions
image-classification-2018-11-28-03-15-55-040	default-variant-Name	ml.m4.xlarge	1	1	<a href="#">Edit   Remove</a>

Add model

Create endpoint configuration



7. Choose **Edit** next to the name of the model and specify the correct **Instance type** on the **Edit Production Variant** page. It is imperative that the **Instance type** value match the one specified in your compilation job.

**Edit Production Variant**

Model name: image-classification-2018-11-28-03-15-55-040

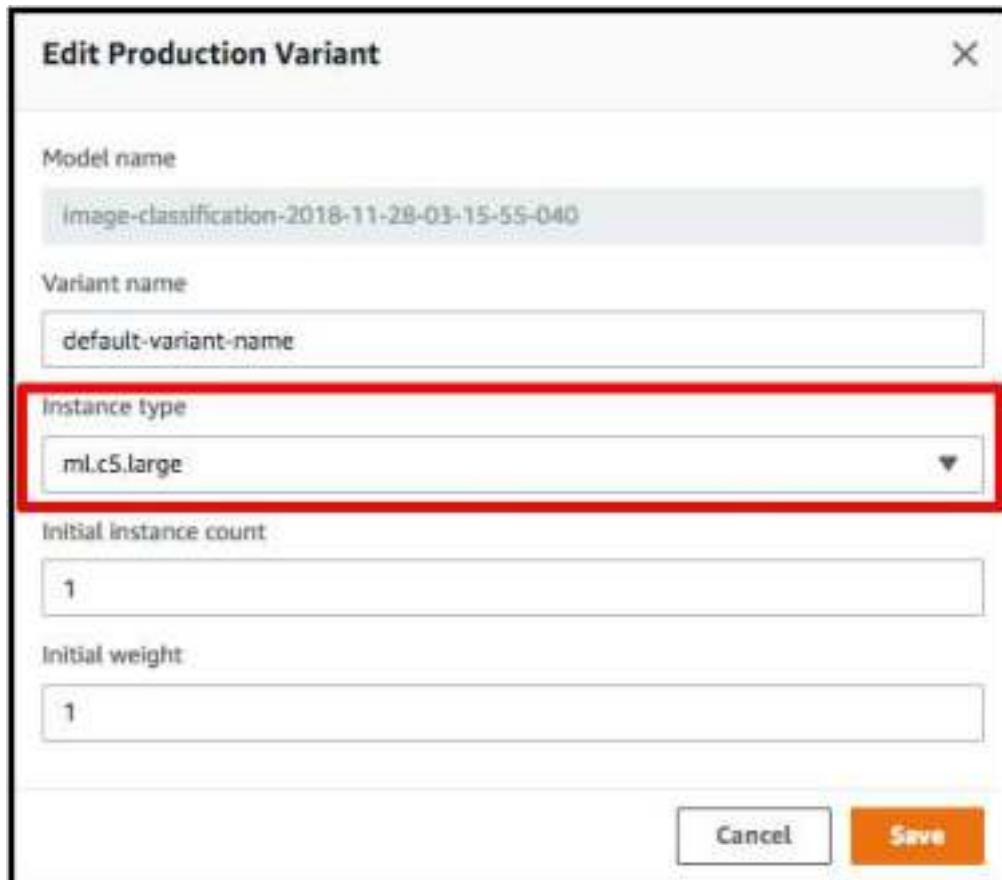
Variant name: default-variant-name

Instance type: ml.c5.large

Initial instance count: 1

Initial weight: 1

**Cancel** **Save**



8. Choose **Save**.
9. On the **New endpoint configuration** page, choose **Create endpoint configuration**, and then choose **Create endpoint**.

## Request Inferences from a Deployed Service

If you have followed instructions in [Deploy a Model \(p. 2039\)](#), you should have a SageMaker endpoint set up and running. Regardless of how you deployed your Neo-compiled model, there are three ways you can submit inference requests:

### Topics

- [Request Inferences from a Deployed Service \(Amazon SageMaker SDK\) \(p. 2057\)](#)
- [Request Inferences from a Deployed Service \(Boto3\) \(p. 2059\)](#)
- [Request Inferences from a Deployed Service \(Amazon CLI\) \(p. 2060\)](#)

### Request Inferences from a Deployed Service (Amazon SageMaker SDK)

Use the following the code examples to request inferences from your deployed service based on the framework you used to train your model. The code examples for the different frameworks are similar. The main difference is that TensorFlow requires `application/json` as the content type.

## PyTorch and MXNet

If you are using **PyTorch v1.4 or later** or **MXNet 1.7.0 or later** and you have an Amazon SageMaker endpoint InService, you can make inference requests using the `predictor` package of the SageMaker SDK for Python.

### Note

The API varies based on the SageMaker SDK for Python version:

- For version 1.x, use the `RealTimePredictor` and `Predict` API.
- For version 2.x, use the `Predictor` and the `Predict` API.

The following code example shows how to use these APIs to send an image for inference:

### SageMaker Python SDK v1.x

```
from sagemaker.predictor import RealTimePredictor
endpoint = 'insert name of your endpoint here'

# Read image into memory
payload = None
with open("image.jpg", 'rb') as f:
    payload = f.read()

predictor = RealTimePredictor(endpoint=endpoint, content_type='application/x-image')
inference_response = predictor.predict(data=payload)
print (inference_response)
```

### SageMaker Python SDK v2.x

```
from sagemaker.predictor import Predictor
endpoint = 'insert name of your endpoint here'

# Read image into memory
payload = None
with open("image.jpg", 'rb') as f:
    payload = f.read()

predictor = Predictor(endpoint)
inference_response = predictor.predict(data=payload)
print (inference_response)
```

## TensorFlow

The following code example shows how to use the SageMaker Python SDK API to send an image for inference:

```
from sagemaker.predictor import Predictor
from PIL import Image
import numpy as np
import json

endpoint = 'insert the name of your endpoint here'

# Read image into memory
image = Image.open(input_file)
batch_size = 1
image = np.asarray(image.resize((224, 224)))
```

```

image = image / 128 - 1
image = np.concatenate([image[np.newaxis, :, :]] * batch_size)
body = json.dumps({"instances": image.tolist()})

predictor = Predictor(endpoint)
inference_response = predictor.predict(data=body)
print(inference_response)

```

## Request Inferences from a Deployed Service (Boto3)

You can submit inference requests using SageMaker SDK for Python (Boto3) client and [invoke\\_endpoint\(\)](#) API once you have an SageMaker endpoint InService. The following code example shows how to send an image for inference:

PyTorch and MXNet

```

import boto3

import json

endpoint = 'insert name of your endpoint here'

runtime = boto3.Session().client('sagemaker-runtime')

# Read image into memory
with open(image, 'rb') as f:
    payload = f.read()
# Send image via InvokeEndpoint API
response = runtime.invoke_endpoint(EndpointName=endpoint, ContentType='application/x-
image', Body=payload)

# Unpack response
result = json.loads(response['Body'].read().decode())

```

TensorFlow

For TensorFlow submit an input with application/json for the content type.

```

from PIL import Image
import numpy as np
import json
import boto3

client = boto3.client('sagemaker-runtime')
input_file = 'path/to/image'
image = Image.open(input_file)
batch_size = 1
image = np.asarray(image.resize((224, 224)))
image = image / 128 - 1
image = np.concatenate([image[np.newaxis, :, :]] * batch_size)
body = json.dumps({"instances": image.tolist()})
ioc_predictor_endpoint_name = 'insert name of your endpoint here'
content_type = 'application/json'
ioc_response = client.invoke_endpoint(
    EndpointName=ioc_predictor_endpoint_name,
    Body=body,
    ContentType=content_type
)

```

XGBoost

For an XGBoost application, you should submit a CSV text instead:

```
import boto3
import json

endpoint = 'insert your endpoint name here'

runtime = boto3.Session().client('sagemaker-runtime')

csv_text = '1,-1.0,1.0,1.5,2.6'
# Send CSV text via InvokeEndpoint API
response = runtime.invoke_endpoint(EndpointName=endpoint, ContentType='text/csv',
    Body=csv_text)
# Unpack response
result = json.loads(response['Body'].read().decode())
```

Note that BYOM allows for a custom content type. For more information, see [runtime\\_InvokeEndpoint](#).

## Request Inferences from a Deployed Service (Amazon CLI)

Inference requests can be made with the `sagemaker-runtime invoke-endpoint` once you have an Amazon SageMaker endpoint InService. You can make inference requests with the Amazon Command Line Interface (Amazon CLI). The following example shows how to send an image for inference:

```
aws sagemaker-runtime invoke-endpoint --endpoint-name 'insert name of your endpoint here'
--body fileb://image.jpg --content-type=application/x-image output_file.txt
```

An `output_file.txt` with information about your inference requests is made if the inference was successful.

For TensorFlow submit an input with `application/json` as the content type.

```
aws sagemaker-runtime invoke-endpoint --endpoint-name 'insert name of your endpoint here'
--body fileb://input.json --content-type=application/json output_file.txt
```

## Inference Container Images

Based on your use case, replace the highlighted portion in the inference image URI template provided below with appropriate values.

### Amazon SageMaker XGBoost

```
aws_account_id.dkr.ecr.aws_region.amazonaws.com/xgboost-neo:latest
```

Replace `aws_account_id` from the table at the end of this page based on the `aws_region` you used.

### TensorFlow

CPU or GPU instance types

```
aws_account_id.dkr.ecr.aws_region.amazonaws.com/sagemaker-inference-
tensorflow:fx_version-instance_type-py3
```

Replace `aws_account_id` from the table at the end of this page based on the `aws_region` you used.

Replace `fx_version` with 1.15.3.

Replace `instance_type` with either cpu or gpu.

Inferentia instance types

```
aws_account_id.dkr.ecr.aws_region.amazonaws.com/sagemaker-neo-
tensorflow:fx_version-instance_type-py3
```

Replace `aws_account_id` from the table at the end of this page based on the `aws_region` you used. Note that for instance type inf only us-east-1 and us-west-2 are supported.

Replace `fx_version` with 1.15.0

Replace `instance_type` with inf.

## MXNet

CPU or GPU instance types

```
aws_account_id.dkr.ecr.aws_region.amazonaws.com/sagemaker-inference-
mxnet:fx_version-instance_type-py3
```

Replace `aws_account_id` from the table at the end of this page based on the `aws_region` you used.

Replace `fx_version` with 1.8.0.

Replace `instance_type` with either cpu or gpu.

Inferentia instance types

```
aws_account_id.dkr.ecr.aws_region.amazonaws.com/sagemaker-neo-
mxnet:fx_version-instance_type-py3
```

Replace `aws_region` with either us-east-1 or us-west-2.

Replace `aws_account_id` from the table at the end of this page based on the `aws_region` you used.

Replace `fx_version` with 1.5.1.

Replace `instance_type` with inf.

## PyTorch

CPU or GPU instance types

```
aws_account_id.dkr.ecr.aws_region.amazonaws.com/sagemaker-inference-
pytorch:fx_version-instance_type-py3
```

Replace `aws_account_id` from the table at the end of this page based on the `aws_region` you used.

Replace `fx_version` with 1.4, 1.5, or 1.6.

Replace `instance_type` with either cpu or gpu.

Inferentia instance types

```
aws_account_id.dkr.ecr.aws_region.amazonaws.com/sagemaker-neo-
pytorch:fx_version-instance_type-py3
```

Replace `aws_region` with either `us-east-1` or `us-west-2`.

Replace `aws_account_id` from the table at the end of this page based on the `aws_region` you used.

Replace `fx_version` with `1.5.1`.

Replace `instance_type` with `inf`.

## Darknet

`aws_account_id.dkr.ecr.aws_region.amazonaws.com/sagemaker-neo-darknet:instance-type`

Replace `aws_account_id` from the table at the end of this page based on the `aws_region` you used.

Replace `instance-type` with either `cpu` or `gpu`.

The following table maps `aws_account_id` with `aws_region`. Use this table to find the correct inference image URI you need for your application.

<code>aws_account_id</code>	<code>aws_region</code>
785573368785	us-east-1
007439368137	us-east-2
710691900526	us-west-1
301217895009	us-west-2
802834080501	eu-west-1
205493899709	eu-west-2
254080097072	eu-west-3
601324751636	eu-north-1
966458181534	eu-south-1
746233611703	eu-central-1
110948597952	ap-east-1
763008648453	ap-south-1
941853720454	ap-northeast-1
151534178276	ap-northeast-2
324986816169	ap-southeast-1
355873309152	ap-southeast-2
474822919863	cn-northwest-1
472730292857	cn-north-1
756306329178	sa-east-1

aws_account_id	aws_region
464438896020	ca-central-1
836785723513	me-south-1
774647643957	af-south-1

## Edge Devices

Amazon SageMaker Neo provides compilation support for popular machine learning frameworks. You can deploy your Neo-compiled edge devices such as the Raspberry Pi 3, Texas Instruments' Sitara, Jetson TX1, and more. For a full list of supported frameworks and edge devices, see [Supported Frameworks, Devices, Systems, and Architectures](#).

You must configure your edge device so that it can use Amazon services. One way to do this is to install DLR and Boto3 to your device. To do this, you must set up the authentication credentials. See [Boto3 Amazon Configuration](#) for more information. Once your model is compiled and your edge device is configured, you can download the model from Amazon S3 to your edge device. From there, you can use the [Deep Learning Runtime \(DLR\)](#) to read the compiled model and make inferences.

For first-time users, we recommend you check out the [Getting Started](#) guide. This guide walks you through how to set up your credentials, compile a model, deploy your model to a Raspberry Pi 3, and make inferences on images.

### Topics

- [Supported Frameworks, Devices, Systems, and Architectures \(p. 2063\)](#)
- [Deploy Models \(p. 2072\)](#)
- [Getting Started with Neo on Edge Devices \(p. 2073\)](#)

## Supported Frameworks, Devices, Systems, and Architectures

Amazon SageMaker Neo supports common machine learning frameworks, edge devices, operating systems, and chip architectures. Find out if Neo supports your framework, edge device, OS, and chip architecture by selecting one of the topics below.

You can find a list of models that have been tested by the Amazon SageMaker Neo Team in the [Tested Models \(p. 2067\)](#) section.

### Note

Ambarella devices require additional files to be included within the compressed TAR file before it is sent for compilation. For more information, see [Troubleshoot Ambarella Errors \(p. 2083\)](#).

### Topics

- [Supported Frameworks \(p. 2063\)](#)
- [Supported Devices, Chip Architectures, and Systems \(p. 2065\)](#)
- [Tested Models \(p. 2067\)](#)

## Supported Frameworks

Amazon SageMaker Neo supports the following frameworks.

Framework	Framework Version	Model Version	Models	Model Formats (packaged in *.tar.gz)	Toolkits
MXNet	1.6.0	Supports 1.6.0 or earlier	Image Classification, Object Detection, Semantic Segmentation, Pose Estimation, Activity Recognition	One symbol file (.json) and one parameter file (.params)	GluonCV v0.8.0
ONNX	1.5.0	Supports 1.5.0 or earlier	Image Classification, SVM	One model file (.onnx)	
Keras	2.2.4	Supports 2.2.4 or earlier	Image Classification	One model definition file (.h5)	
PyTorch	1.6.0	Supports 1.6.0 or earlier	Image Classification	One model definition file (.pth)	
TensorFlow	1.15.0	Supports 1.15.0 or earlier	Image Classification	*For saved models, one .pb or one .pbtxt file and a variables directory that contains variables *For frozen models, only one .pb or .pbtxt file	
TensorFlow-Lite	1.13.1	Supports 1.13.1 or earlier	Image Classification, Object Detection	One model definition flatbuffer file (.tflite)	
XGBoost	0.9	Supports 0.9 or earlier	Decision Trees	One XGBoost model file (.model) where the number of nodes in a tree is less than $2^{31}$	
DARKNET			Image Classification, Object Detection	One config (.cfg) file and one weights (.weights) file	

## Supported Devices, Chip Architectures, and Systems

Amazon SageMaker Neo supports the following devices, chip architectures, and operating systems.

### Devices

You can select a device using the dropdown list in the [Amazon SageMaker console](#) or by specifying the `TargetDevice` in the output configuration of the [createCompilationJob](#) API.

You can choose from one of the following edge devices:

Device List	System on a Chip (SoC)	Operating System	Architecture	Accelerator	Compiler Options Example
aisage		Linux	ARM64	Mali	
amba_cv22	CV22	Arch Linux	ARM64	cvflow	
coreml		iOS, macOS			{"class_labels": "imagenet_labels_1000..."}
deeplens	Intel Atom	Linux	X86_64	Intel Graphics	
imx8qm	NXP imx8	Linux	ARM64		
jacinto_tda4vm	TDA4VM	Linux	ARM	TDA4VM	
jetson_nano		Linux	ARM64	NVIDIA	{"gpu-code": "sm_53", "trt-ver": "5.0.6", "cuda-ver": "10.0"}
jetson_tx1		Linux	ARM64	NVIDIA	{"gpu-code": "sm_53", "trt-ver": "6.0.1", "cuda-ver": "10.0"}
jetson_tx2		Linux	ARM64	NVIDIA	{"gpu-code": "sm_62", "trt-ver": "6.0.1", "cuda-ver": "10.0"}
jetson_xavier		Linux	ARM64	NVIDIA	{"gpu-code": "sm_72", "trt-ver": "5.1.6", "cuda-ver": "10.0"}

Device List	System on a Chip (SoC)	Operating System	Architecture	Accelerator	Compiler Options Example
qcs605		Android	ARM64	Mali	{ 'ANDROID_PLATFORM': 27 }
qcs603		Android	ARM64	Mali	{ 'ANDROID_PLATFORM': 27 }
rasp3b	ARM A56	Linux	ARM_EABIHF		{ 'mattr': [ '+neon' ] }
rasp4	ARM A72				
rk3288		Linux	ARM_EABIHF	Mali	
rk3399		Linux	ARM64	Mali	
sbe_c		Linux	x86_64		{ 'mcpu': 'core-avx2' }
sitara_am57x	AM57X	Linux	ARM64	EVE and/or C66x DSP	
x86_win32		Windows 10	X86_32		
x86_win64		Windows 10	X86_32		

For more information about JSON key-value compiler options for each target device, see the [CompilerOptions](#) field in the [OutputConfig API](#) data type.

### Systems and Chip Architectures

The following look-up tables provide information regarding available operating systems and architectures for Neo model compilation jobs.

#### Linux

	X86_64	X86	ARM64	ARM_EABIHF	ARM_EABI
No accelerator (CPU)	X		X	X	X
Nvidia GPU	X		X		
Intel_Graphics	X				
ARM Mali			X	X	X

#### Android

	X86_64	X86	ARM64	ARM_EABIHF	ARM_EABI
No accelerator (CPU)	X	X	X		X

	X86_64	X86	ARM64	ARM_EABIHF	ARM_EABI
Nvidia GPU					
Intel_Graphics	X	X			
ARM Mali			X		X

Windows

	X86_64	X86	ARM64	ARM_EABIHF	ARM_EABI
No accelerator (CPU)	X	X			

## Tested Models

The following collapsible sections provide information about machine learning models that were tested by the Amazon SageMaker Neo team. Expand the collapsible section based on your framework to check if a model was tested.

### Note

This is not a comprehensive list of models that can be compiled with Neo.

See [Supported Frameworks \(p. 2063\)](#) and [SageMaker Neo Supported Operators](#) to find out if you can compile your model with SageMaker Neo.

## DarkNet

Models	ARM V8	ARM Mali	Ambare CV22	Nvidia	Panoram	TI TDA4VM	Qualco QCS603	X86_Linu	X86_Wi		
Alexnet											
Resnet50X	X			X	X	X		X	X		
YOLOv2				X	X	X		X	X		
YOLOv2_Xiny	X			X	X	X		X	X		
YOLOv3_416				X	X	X		X	X		
YOLOv3_Xiny	X			X	X	X		X	X		

## MXNet

Models	ARM V8	ARM Mali	Ambarell CV22	Nvidia	Panoram	TI TDA4VM	Qualcom QCS603	X86_Linu	X86_Windows
Alexnet			X						
Densenet121			X						
DenseNet201	X	X	X	X	X	X		X	X
GoogLeNetX	X			X	X	X		X	X

Models	ARM V8	ARM Mali	Ambarell CV22	Nvidia	Panoram	TI TDA4VM	Qualcom QCS603	X86_Linu	X86_Windows
InceptionV3				X	X	X		X	X
MobileNetV2.75	X			X	X	X			X
MobileNetV2.0	X	X		X	X	X			X
MobileNetV2_0.5	X			X	X	X			X
MobileNetV2_1.0	X	X		X	X	X	X	X	X
MobileNetV3_Large	X	X		X	X	X	X	X	X
MobileNetV3_Small	X	X		X	X	X	X	X	X
ResNeSt50				X	X			X	X
ResNet18_V1	X	X		X	X	X			X
ResNet18_V2	X			X	X	X			X
ResNet50_V1	X	X		X	X	X		X	X
ResNet50_V2	X	X		X	X	X		X	X
ResNext101_32x4d									
ResNext50X32x4d		X		X	X			X	X
SENet_154				X	X	X		X	X
SE_ResNext50_32x4d				X	X	X		X	X
SqueezeNetV1.0	X	X		X	X	X			X
SqueezeNetV1.1	X	X		X	X	X		X	X
VGG11	X	X		X	X			X	X
Xception	X	X		X	X	X		X	X
darknet53X	X			X	X	X		X	X
resnet18_v1b_0.89	X			X	X	X			X
resnet50_v1d_0.11	X			X	X	X			X
resnet50_v1d_0.86	X	X		X	X	X		X	X
ssd_512_mobilenet1.0_coco	X			X	X	X		X	X
ssd_512_mobilenet1.0_voc	X			X	X	X		X	X
ssd_resnetX0_v1		X		X	X			X	X
yolo3_darknet53_coco				X	X			X	X
yolo3_mobilenet1.0_Xoco				X	X	X		X	X
deeplab_resnet50			X						

### Keras

Models	ARM V8	ARM Mali	Ambarell CV22	Nvidia	Panoram	TI TDA4VM	Qualcom QCS603	X86_Linu	X86_Windows
densenet1X1	X	X		X	X	X		X	X
densenet2X1	X	X		X	X	X			X
inception_X3	X			X	X	X		X	X
mobilenet_Xv1	X	X		X	X	X		X	X
mobilenet_Xv2	X	X		X	X	X		X	X
resnet152_v1				X	X				X
resnet152_v2				X	X				X
resnet50_v1	X	X		X	X			X	X
resnet50_v2	X	X		X	X	X		X	X
vgg16			X	X	X			X	X

### ONNX

Models	ARM V8	ARM Mali	Ambarell CV22	Nvidia	Panoram	TI TDA4VM	Qualcom QCS603	X86_Linu	X86_Windows
alexnet			X						
mobilenetX2-1.0	X	X	X	X	X	X		X	X
resnet18vX				X	X				X
resnet18vX				X	X				X
resnet50vX		X	X	X				X	X
resnet50vX		X	X	X				X	X
resnet152v1			X	X	X	X			X
resnet152v2			X	X	X	X			X
squeezeX1.1		X	X	X	X	X		X	X
vgg19			X						X

### PyTorch (FP32)

Models	ARM V8	ARM Mali	Ambarell CV22	Nvidia	Panoram	TI TDA4VM	Qualcom QCS603	X86_Linu	X86_Windows
densenet1X1	X			X	X	X		X	X

Models	ARM V8	ARM Mali	Ambarella CV22	Nvidia	Panoram	TI TDA4VM	Qualcom QCS603	X86_Linu	X86_Windows
inception_v3	X			X	X	X		X	X
resnet152				X	X	X			X
resnet18	X	X		X	X	X			X
resnet50	X	X		X	X			X	X
squeezezenet1.0	X			X	X	X			X
squeezezenet1.1	X			X	X	X		X	X

### TensorFlow

#### TensorFlow

Models	ARM V8	ARM Mali	Ambarella CV22	Nvidia	Panorama	TI TDA4VM	Qualcom QCS603	X86_Linux	X86_Windows
densenet201	X	X		X	X	X		X	X
inception_v3	X	X		X	X	X		X	X
mobilenet100_v1	X	X		X	X	X			X
mobilenet100_v2.0	X	X		X	X	X		X	X
mobilenet130_v2	X			X	X	X			X
mobilenet140_v2	X	X		X	X	X		X	X
resnet50_v1.5	X			X	X	X		X	X
resnet50_v2	X	X		X	X	X		X	X
squeezezenet	X	X		X	X	X		X	X

#### TensorFlow.Keras

Models	ARM V8	ARM Mali	Ambarella CV22	Nvidia	Panorama	TI TDA4VM	Qualcom QCS603	X86_Linux	X86_Windows
DenseNet121	X			X	X	X		X	X
DenseNet201	X			X	X	X			X
InceptionV3	X			X	X	X		X	X
MobileNet	X	X		X	X	X		X	X
MobileNetv2	X			X	X	X		X	X
NASNetLarge				X	X			X	X
NASNetMobile	X			X	X	X		X	X

Models	ARM V8	ARM Mali	Ambarella CV22	Nvidia	Panorama	TI TDA4VM	Qualcomm QCS603	X86_Linux	X86_Windows
ResNet101				X	X	X			X
ResNet101V2				X	X	X			X
ResNet152				X	X				X
ResNet152v2				X	X				X
ResNet50	X	X		X	X			X	X
ResNet50V2	X			X	X	X		X	X
VGG16				X	X			X	X
Xception	X	X		X	X	X		X	X

### TensorFlow-Lite

TensorFlow-Lite (FP32)

Models	ARM V8	ARM Mali	Ambarella CV22	Nvidia	Panorama	TI TDA4VM	Qualcomm QCS603	X86_Linux	X86_Windows
densenet_2018_04_27				X	X	X			X
inception_resnet_v2_2018_04_27				X	X	X			X
inception_v3_2018_04_27				X	X	X			X
inception_v4_2018_04_27				X	X	X			X
mnasnet_05_224_09_07_2018				X	X	X			X
mnasnet_10_224_09_07_2018				X	X	X			X
mnasnet_13_224_09_07_2018				X	X	X			X
mobilenetXv1_0.25_128				X	X	X			X
mobilenetXv1_0.25_224				X	X	X			X
mobilenetXv1_0.5_128				X	X	X			X
mobilenetXv1_0.5_224				X	X	X			X
mobilenetXv1_0.75_128				X	X	X			X
mobilenetXv1_0.75_224				X	X	X			X
mobilenetXv1_1.0_128				X	X	X			X
mobilenetXv1_1.0_192				X	X	X			X
mobilenetXv2_1.0_224				X	X	X			X
resnet_v2_101				X	X	X			X
squeezeNext_2018_04_27				X	X	X			X

### TensorFlow-Lite (INT8)

Models	ARM V8	ARM Mali	Ambarel CV22	Nvidia	Panoram	TI TDA4VM	Qualcom QCS603	X86_Linu	X86_Windows
inception_v1							X		
inception_v2							X		
inception_X3						X	X		X
inception_X4_299						X	X		X
mobilenetXv1_0.25_128						X			X
mobilenetXv1_0.25_224						X			X
mobilenetXv1_0.5_128						X			X
mobilenetXv1_0.5_224						X			X
mobilenetXv1_0.75_128						X			X
mobilenetXv1_0.75_224						X	X		X
mobilenetXv1_1.0_128						X			X
mobilenetXv1_1.0_224						X	X		X
mobilenetXv2_1.0_224						X	X		X
deeplab-v3_513							X		

## Deploy Models

You can deploy the compute module to resource-constrained edge devices by: downloading the compiled model from Amazon S3 to your device and using [DLR](#), or you can use [Amazon IoT Greengrass](#).

Before moving on, make sure your edge device must be supported by SageMaker Neo. See, [Supported Frameworks, Devices, Systems, and Architectures](#) to find out what edge devices are supported. Make sure that you specified your target edge device when you submitted the compilation job, see [Use Neo to Compile a Model](#).

### Deploy a Compiled Model (DLR)

[DLR](#) is a compact, common runtime for deep learning models and decision tree models. DLR uses the [TVM](#) runtime, [Treelite](#) runtime, NVIDIA TensorRT™, and can include other hardware-specific runtimes. DLR provides unified Python/C++ APIs for loading and running compiled models on various devices.

You can install latest release of DLR package using the following pip command:

```
pip install dlr
```

For installation of DLR on GPU targets or non-x86 edge devices, please refer to [Releases](#) for prebuilt binaries, or [Installing DLR](#) for building DLR from source. For example, to install DLR for Raspberry Pi 3, you can use:

```
pip install https://neo-ai-dlr-release.s3-us-west-2.amazonaws.com/v1.3.0/pi-armv7l-raspbian4.14.71-glibc2_24-libstdcpp3_4/dlr-1.3.0-py3-none-any.whl
```

## Deploy a Model (Amazon IoT Greengrass)

Amazon IoT Greengrass extends cloud capabilities to local devices. It enables devices to collect and analyze data closer to the source of information, react autonomously to local events, and communicate securely with each other on local networks. With Amazon IoT Greengrass, you can perform machine learning inference at the edge on locally generated data using cloud-trained models. Currently, you can deploy models on to all Amazon IoT Greengrass devices based on ARM Cortex-A, Intel Atom, and Nvidia Jetson series processors. For more information on deploying a Lambda inference application to perform machine learning inferences with Amazon IoT Greengrass, see [How to configure optimized machine learning inference using the Amazon Management Console](#).

## Getting Started with Neo on Edge Devices

This guide to getting started with Amazon SageMaker Neo shows you how to compile a model, set up your device, and make inferences on your device. Most of the code examples use Boto3. We provide commands using Amazon CLI where applicable, as well as instructions on how to satisfy prerequisites for Neo.

### Note

You can run the following code snippets on your local machine, within a SageMaker notebook, within SageMaker Studio, or (depending on your edge device) on your edge device. The setup is similar; however, there are two main exceptions if you run this guide within a SageMaker notebook instance or SageMaker Studio session:

- You do not need to install Boto3.
- You do not need to add the ‘AmazonSageMakerFullAccess’ IAM policy

This guide assumes you are running the following instructions on your edge device.

## Prerequisites

### 1. Install Boto3

If you are running these commands on your edge device, you must install the Amazon SDK for Python (Boto3). Within a Python environment (preferably a virtual environment), run the following locally on your edge device’s terminal or within a Jupyter notebook instance:

Terminal

```
$ pip install boto3
```

Jupyter Notebook

```
!pip install boto3
```

### 2. Set Up Amazon Credentials

You need to set up Amazon Web Services credentials on your device in order to run SDK for Python (Boto3). By default, the Amazon credentials should be stored in the file `~/.aws/credentials` on your edge device. Within the credentials file, you should see two environment variables: `aws_access_key_id` and `aws_secret_access_key`.

In your terminal, run:

```
$ more ~/.aws/credentials

[default]
aws_access_key_id = YOUR_ACCESS_KEY
aws_secret_access_key = YOUR_SECRET_KEY
```

The [Amazon General Reference Guide](#) has instructions on how to get the necessary `aws_access_key_id` and `aws_secret_access_key`. For more information on how to set up credentials on your device, see the [Boto3](#) documentation.

### 3. Set up an IAM Role and attach policies.

Neo needs access to your S3 bucket URI. Create an IAM role that can run SageMaker and has permission to access the S3 URI. You can create an IAM role either by using SDK for Python (Boto3), the console, or the Amazon CLI. The following example illustrates how to create an IAM role using SDK for Python (Boto3):

```
import boto3

AWS_REGION = 'aws-region'

# Create an IAM client to interact with IAM
iam_client = boto3.client('iam', region_name=AWS_REGION)
role_name = 'role-name'
```

For more information on how to create an IAM role with the console, Amazon CLI, or through the Amazon API, see [Creating an IAM user in your Amazon account](#).

Create a dictionary describing the IAM policy you are attaching. This policy is used to create a new IAM role.

```
policy = {
    'Statement': [
        {
            'Action': 'sts:AssumeRole',
            'Effect': 'Allow',
            'Principal': {'Service': 'sagemaker.amazonaws.com'},
        },
        'Version': '2012-10-17'
    ]
}
```

Create a new IAM role using the policy you defined above:

```
import json

new_role = iam_client.create_role(
    AssumeRolePolicyDocument=json.dumps(policy),
    Path='/',
    RoleName=role_name
)
```

You need to know what your Amazon Resource Name (ARN) is when you create a compilation job in a later step, so store it in a variable as well.

```
role_arn = new_role['Role']['Arn']
```

Now that you have created a new role, attach the permissions it needs to interact with Amazon SageMaker and Amazon S3:

```
iam_client.attach_role_policy(
    RoleName=role_name,
    PolicyArn='arn:aws:iam::aws:policy/AmazonSageMakerFullAccess'
)

iam_client.attach_role_policy(
    RoleName=role_name,
    PolicyArn='arn:aws:iam::aws:policy/AmazonS3FullAccess'
);
```

#### 4. Create an Amazon S3 bucket to store your model artifacts

SageMaker Neo will access your model artifacts from Amazon S3

Boto3

```
# Create an S3 client
s3_client = boto3.client('s3', region_name=AWS_REGION)

# Name buckets
bucket='name-of-your-bucket'

# Check if bucket exists
if boto3.resource('s3').Bucket(bucket) not in boto3.resource('s3').buckets.all():
    s3_client.create_bucket(
        Bucket=bucket,
        CreateBucketConfiguration={
            'LocationConstraint': AWS_REGION
        }
)
else:
    print(f'Bucket {bucket} already exists. No action needed.')
```

CLI

```
$ aws s3 mb s3://'name-of-your-bucket' --region specify-your-region

# Check your bucket exists
$ aws s3 ls s3://'name-of-your-bucket'/
```

#### 5. Train a machine learning model

See [Train a Model with Amazon SageMaker](#) for more information on how to train a machine learning model using Amazon SageMaker. You can optionally upload your locally trained model directly into an Amazon S3 URI bucket.

**Note**

Make sure the model is correctly formatted depending on the framework you used. See [What input data shapes does SageMaker Neo expect?](#)

If you do not have a model yet, use the `curl` command to get a local copy of the `coco_ssd_mobilenet` model from TensorFlow's website. The model you just copied is an object detection model trained from the [COCO dataset](#). Type the following into your Jupyter notebook:

```
model_zip_filename = './coco_ssd_mobilenet_v1_1.0.zip'
!curl http://storage.googleapis.com/download.tensorflow.org/models/tflite/
coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip \
```

```
--output {model_zip_filename}
```

Note that this particular example was packaged in a .zip file. Unzip this file and repackage it as a compressed tarfile (.tar.gz) before using it in later steps. Type the following into your Jupyter notebook:

```
# Extract model from zip file
!unzip -u {model_zip_filename}

model_filename = 'detect.tflite'
model_name = model_filename.split('.')[0]

# Compress model into .tar.gz so SageMaker Neo can use it
model_tar = model_name + '.tar.gz'
!tar -czf {model_tar} {model_filename}
```

## 6. Upload trained model to an S3 bucket

Once you have trained your machine learning mode, store it in an S3 bucket.

Boto3

```
# Upload model
s3_client.upload_file(filename=model_filename, bucket=bucket, key=model_filename)
```

CLI

Replace `your-model-filename` and `your-S3-bucket` with the name of your S3 bucket.

```
$ aws s3 cp your-model-filename s3://your-S3-bucket
```

## Step 1: Compile the Model

Once you have satisfied the [Prerequisites](#), you can compile your model with Amazon SageMaker Neo. You can compile your model using the Amazon CLI, the console or the [Amazon Web Services SDK for Python \(Boto3\)](#), see [Use Neo to Compile a Model](#). In this example, you will compile your model with Boto3.

To compile a model, SageMaker Neo requires the following information:

### 1. The Amazon S3 bucket URI where you stored the trained model.

If you followed the prerequisites, the name of your bucket is stored in a variable named `bucket`. The following code snippet shows how to list all of your buckets using the Amazon CLI:

```
$ aws s3 ls
```

For example:

```
$ aws s3 ls
2020-11-02 17:08:50 bucket
```

### 2. The Amazon S3 bucket URI where you want to save the compiled model.

The code snippet below concatenates your Amazon S3 bucket URI with the name of an output directory called `output`:

```
s3_output_location = f's3://{bucket}/output'
```

**3. The machine learning framework you used to train your model.**

Define the framework you used to train your model.

```
framework = 'framework-name'
```

For example, if you wanted to compile a model that was trained using TensorFlow, you could either use `tflite` or `tensorflow`. Use `tflite` if you want to use a lighter version of TensorFlow that uses less storage memory.

```
framework = 'tflite'
```

For a complete list of Neo-supported frameworks, see [Supported Frameworks, Devices, Systems, and Architectures](#).

**4. The shape of your model's input.**

Neo requires the name and shape of your input tensor. The name and shape are passed in as key-value pairs. `value` is a list of the integer dimensions of an input tensor and `key` is the exact name of an input tensor in the model.

```
data_shape = '{"name": [tensor-shape]}'
```

For example:

```
data_shape = '{"normalized_input_image_tensor": [1, 300, 300, 3]}'
```

**Note**

Make sure the model is correctly formatted depending on the framework you used. See [What input data shapes does SageMaker Neo expect?](#) The key in this dictionary must be changed to the new input tensor's name.

**5. Either the name of the target device to compile for or the general details of the hardware platform**

```
target_device = 'target-device-name'
```

For example, if you want to deploy to a Raspberry Pi 3, use:

```
target_device = 'rasp3b'
```

You can find the entire list of supported edge devices in [Supported Frameworks, Devices, Systems, and Architectures](#).

Now that you have completed the previous steps, you can submit a compilation job to Neo.

```
# Create a SageMaker client so you can submit a compilation job
sagemaker_client = boto3.client('sagemaker', region_name=AWS_REGION)

# Give your compilation job a name
compilation_job_name = 'getting-started-demo'
print(f'Compilation job for {compilation_job_name} started')
```

```

response = sagemaker_client.create_compilation_job(
    CompilationJobName=compilation_job_name,
    RoleArn=role_arn,
    InputConfig={
        'S3Uri': s3_input_location,
        'DataInputConfig': data_shape,
        'Framework': framework.upper()
    },
    OutputConfig={
        'S3OutputLocation': s3_output_location,
        'TargetDevice': target_device
    },
    StoppingCondition={
        'MaxRuntimeInSeconds': 900
    }
)

# Optional - Poll every 30 sec to check completion status
import time

while True:
    response =
    sagemaker_client.describe_compilation_job(CompilationJobName=compilation_job_name)
    if response['CompilationJobStatus'] == 'COMPLETED':
        break
    elif response['CompilationJobStatus'] == 'FAILED':
        raise RuntimeError('Compilation failed')
    print('Compiling ...')
    time.sleep(30)
print('Done!')

```

If you want additional information for debugging, include the following print statement:

```
print(response)
```

If the compilation job is successful, your compiled model is stored in the output Amazon S3 bucket you specified earlier (`s3_output_location`). Download your compiled model locally:

```

object_path = f'output/{model}-{target_device}.tar.gz'
neo_compiled_model = f'compiled-{model}.tar.gz'
s3_client.download_file(bucket, object_path, neo_compiled_model)

```

## Step 2: Set Up Your Device

You will need to install packages on your edge device so that your device can make inferences. You will also need to either install [Amazon IoT Greengrass](#) core or [Deep Learning Runtime \(DLR\)](#). In this example, you will install packages required to make inferences for the `coco_ssd_mobilenet` object detection algorithm and you will use DLR.

### 1. Install additional packages

In addition to Boto3, you must install certain libraries on your edge device. What libraries you install depends on your use case.

For example, for the `coco_ssd_mobilenet` object detection algorithm you downloaded earlier, you need to install [NumPy](#) for data manipulation and statistics, [PIL](#) to load images, and [Matplotlib](#) to generate plots. You also need a copy of TensorFlow if you want to gauge the impact of compiling with Neo versus a baseline.

```
!pip3 install numpy pillow tensorflow matplotlib
```

## 2. Install inference engine on your device

To run your Neo-compiled model, install the [Deep Learning Runtime \(DLR\)](#) on your device. DLR is a compact, common runtime for deep learning models and decision tree models. On x86\_64 CPU targets running Linux, you can install the latest release of the DLR package using the following pip command:

```
!pip install dlr
```

For installation of DLR on GPU targets or non-x86 edge devices, refer to [Releases](#) for prebuilt binaries, or [Installing DLR](#) for building DLR from source. For example, to install DLR for Raspberry Pi 3, you can use:

```
!pip install https://neo-ai-dlr-release.s3-us-west-2.amazonaws.com/v1.3.0/pi-armv7l-raspbian4.14.71-glibc2_24-libstdcpp3_4/dlr-1.3.0-py3-none-any.whl
```

## Step 3: Make Inferences on Your Device

In this example, you will use Boto3 to download the output of your compilation job onto your edge device. You will then import DLR, download an example images from the dataset, resize this image to match the model's original input, and then you will make a prediction.

### 1. Download your compiled model from Amazon S3 to your device and extract it from the compressed tarfile.

```
# Download compiled model locally to edge device
object_path = f'output/{model_name}-{target_device}.tar.gz'
neo_compiled_model = f'compiled-{model_name}.tar.gz'
s3_client.download_file(bucket_name, object_path, neo_compiled_model)

# Extract model from .tar.gz so DLR can use it
!mkdir ./dlr_model # make a directory to store your model (optional)
!tar -xvf ./compiled-detect.tar.gz --directory ./dlr_model
```

### 2. Import DLR and an initialized DLRModel object.

```
import dlr

device = 'cpu'
model = dlr.DLRModel('./dlr_model', device)
```

### 3. Download an image for inferencing and format it based on how your model was trained.

For the coco\_ssd\_mobilenet example, you can download an image from the [COCO dataset](#) and then reform the image to 300x300:

```
from PIL import Image

# Download an image for model to make a prediction
input_image_filename = './input_image.jpg'
!curl https://farm9.staticflickr.com/8325/8077197378_79efb4805e_z.jpg --output {input_image_filename}

# Format image so model can make predictions
resized_image = image.resize((300, 300))
```

```
# Model is quantized, so convert the image to uint8
x = np.array(resized_image).astype('uint8')
```

#### 4. Use DLR to make inferences.

Finally, you can use DLR to make a prediction on the image you just downloaded:

```
out = model.run(x)
```

For more examples using DLR to make inferences from a Neo-compiled model on an edge device, see the [neo-ai-dlr Github repository](#).

## Troubleshoot Errors

This section contains information about how to understand and prevent common errors, the error messages they generate, and guidance on how to resolve these errors. Before moving on, ask yourself the following questions:

**Did you encounter an error before you deployed your model?** If yes, see [Troubleshoot Neo Compilation Errors](#).

**Did you encounter an error after you compiled your model?** If yes, see [Troubleshoot Neo Inference Errors](#).

**Did you encounter an error trying to compile your model for Ambarella devices?** If yes, see [Troubleshoot Ambarella Errors \(p. 2083\)](#).

## Error Classification Types

This list classifies the *user errors* you can receive from Neo. These include access and permission errors and load errors for each of the supported frameworks. All other errors are *system errors*.

### Client permission error

Neo passes the errors for these straight through from the dependent service.

- *Access Denied* when calling sts:AssumeRole
- Any 400 error when calling Amazon S3 to download or upload a client model
- *PassRole* error

### Load error

Assuming that the Neo compiler successfully loaded .tar.gz from Amazon S3, check whether the tarball contains the necessary files for compilation. The checking criteria is framework-specific:

- **TensorFlow**: Expects only protobuf file (\*.pb or \*.pbtxt). For saved models, expects one variables folder.
- **Pytorch**: Expect only one pytorch file (\*.pth).
- **MXNET**: Expect only one symbol file (\*.json) and one parameter file (\*.params).
- **XGBoost**: Expect only one XGBoost model file (\*.model). The input model has size limitation.

### Compilation error

Assuming that the Neo compiler successfully loaded .tar.gz from Amazon S3, and that the tarball contains necessary files for compilation. The checking criteria is:

- **OperatorNotImplemented:** An operator has not been implemented.
- **OperatorAttributeNotImplemented:** The attribute in the specified operator has not been implemented.
- **OperatorAttributeRequired:** An attribute is required for an internal symbol graph, but it is not listed in the user input model graph.
- **OperatorAttributeValueNotValid:** The value of the attribute in the specific operator is not valid.

#### Topics

- [Troubleshoot Neo Compilation Errors \(p. 2081\)](#)
- [Troubleshoot Neo Inference Errors \(p. 2082\)](#)
- [Troubleshoot Ambarella Errors \(p. 2083\)](#)

## Troubleshoot Neo Compilation Errors

This section contains information about how to understand and prevent common compilation errors, the error messages they generate, and guidance on how to resolve these errors.

#### Topics

- [How to Use This Page \(p. 2081\)](#)
- [Framework-Related Errors \(p. 2081\)](#)
- [Infrastructure-Related Errors \(p. 2082\)](#)

### How to Use This Page

Attempt to resolve your error by going through these sections in the following order:

1. Check that the input of your compilation job satisfies the input requirements. See [What input data shapes does SageMaker Neo expect?](#)
2. Check common [framework-specific errors](#).
3. Check if your error is an [infrastructure error](#).

### Framework-Related Errors

#### TensorFlow

Error	Solution
InputConfiguration: Exactly one .pb file is allowed for TensorFlow models.	Make sure you only provide one .pb or .pbtxt file.
InputConfiguration: Exactly one .pb or .pbtxt file is allowed for TensorFlow models.	Make sure you only provide one .pb or .pbtxt file.
ClientError: InputConfiguration: TVM cannot convert <model zoo> model. Please make sure the framework you selected is correct. The following operators are not implemented: {<operator name>}	Check the operator you chose is supported. See <a href="#">SageMaker Neo Supported Frameworks and Operators</a> .

## Keras

Error	Solution
<code>InputConfiguration: No h5 file provided in &lt;model path&gt;</code>	Check your h5 file is in the Amazon S3 URI you specified. <i>Or</i> Check that the <a href="#">h5 file is correctly formatted</a> .
<code>InputConfiguration: Multiple h5 files provided, &lt;model path&gt;, when only one is allowed</code>	Check you are only providing one h5 file.
<code>ClientError: InputConfiguration: Unable to load provided Keras model. Error: 'sample_weight_mode'</code>	Check the Keras version you specified is supported. See, supported frameworks for <a href="#">cloud instances</a> and <a href="#">edge devices</a> .
<code>ClientError: InputConfiguration: Input input has wrong shape in Input Shape dictionary. Input shapes should be provided in NCHW format.</code>	Check that your model input follows NCHW format. See <a href="#">What input data shapes does SageMaker Neo expect?</a>

## MXNet

Error	Solution
<code>ClientError: InputConfiguration: Only one parameter file is allowed for MXNet model. Please make sure the framework you select is correct.</code>	SageMaker Neo will select the first parameter file given for compilation.

## Infrastructure-Related Errors

Error	Solution
<code>ClientError: InputConfiguration: S3 object does not exist. Bucket: &lt;bucket&gt;, Key: &lt;bucket key&gt;</code>	Check the Amazon S3 URI you provided.
<code>ClientError: InputConfiguration: Bucket &lt;bucket name&gt; is in region &lt;region name&gt; which is different from Amazon Sagemaker service region &lt;service region&gt;</code>	Create an Amazon S3 bucket that is in the same region as the service.
<code>ClientError: InputConfiguration: Unable to untar input model. Please confirm the model is a tar.gz file</code>	Check that your model in Amazon S3 is compressed into a tar.gz file.

## Troubleshoot Neo Inference Errors

This section contains information about how to prevent and resolve some of the common errors you might encounter upon deploying and/or invoking the endpoint. This section applies to **PyTorch 1.4.0 or later** and **MXNet v1.7.0 or later**.

- Make sure the first inference (warm-up inference) on a valid input data is done in `model_fn()`, otherwise the following error message may be seen on the terminal when [predict API](#) is called:

```
An error occurred (ModelError) when calling the InvokeEndpoint operation: Received server error (0) from <users-sagemaker-endpoint> with message "Your invocation timed out while waiting for a response from container model. Review the latency metrics for each container in Amazon CloudWatch, resolve the issue, and try again."
```

- Make sure that the environment variables in the following table are set. If they are not set, the following error message might show up:

**On the terminal:**

```
An error occurred (ModelError) when calling the InvokeEndpoint operation: Received server error (503) from <users-sagemaker-endpoint> with message "{ "code": 503, "type": "InternalServerException", "message": "Prediction failed" }".
```

**In CloudWatch:**

```
W-9001-model-stdout com.amazonaws.ml.mms.wlm.WorkerLifeCycle - AttributeError: 'NoneType' object has no attribute 'transform'
```

Key	Value
SAGEMAKER_PROGRAM	inference.py
SAGEMAKER_SUBMIT_DIRECTORY	/opt/ml/model/code
SAGEMAKER_CONTAINER_LOG_LEVEL	20
SAGEMAKER_REGION	<your region>

- Make sure that the `MMS_DEFAULT_RESPONSE_TIMEOUT` environment variable is set to 500 or a higher value while creating the Amazon SageMaker model; otherwise, the following error message may be seen on the terminal:

```
An error occurred (ModelError) when calling the InvokeEndpoint operation: Received server error (0) from <users-sagemaker-endpoint> with message "Your invocation timed out while waiting for a response from container model. Review the latency metrics for each container in Amazon CloudWatch, resolve the issue, and try again."
```

## Troubleshoot Ambarella Errors

SageMaker Neo requires models to be packaged in a compressed TAR file (\*.tar.gz). Ambarella devices require additional files to be included within the compressed TAR file before it is sent for compilation. Include the following files within your compressed TAR file if you want to compile a model for Ambarella targets with SageMaker Neo:

- A trained model using a framework supported by SageMaker Neo
- A JSON configuration file
- Calibration images

For example, your directory should look similar to the following example:

```
###Mobilenet_v1_1.0_224
```

```

|     ###amba_config.json
|     ###calib_data
|     |     ### data1
|     |     ### data2
|     .
|     .
|     .
|     |     ### data500
|     |     ###mobilenet_v1_1.0_0224_frozen.pb
|

```

The directory is configured as follows:

- `Mobilenet_v1_1.0_224` : The name of file directory that is compressed as a TAR file
- `amba_config.json` : Configuration file
- `calib_data` : Folder containing calibration images
- `mobilenet_v1_1.0_0224_frozen.pb` : TensorFlow model saved as a frozen graph

For information about frameworks supported by SageMaker Neo, see [Supported Frameworks \(p. 2063\)](#).

## Setting up the Configuration File

The configuration file provides information required by the Ambarella toolchain to compile the model. The configuration file must be saved as a JSON file and the name of the file must end with `*config.json`. The following chart shows the contents of the configuration file.

Key	Description	Example
<code>inputs</code>	Dictionary mapping input layers to attribute.	<code>{inputs:{ "data":{...}, "data1":{...} }}</code>
<code>"data"</code>	Input layer name. Note: <code>"data"</code> is an example of the name you can use to label the input layer.	<code>"data"</code>
<code>shape</code>	Describes the shape of the input to the model. This follows the same conventions that SageMaker Neo uses.	<code>"shape": "1,3,224,224"</code>
<code>filepath</code>	Relative path to the directory containing calibration images. These can be binary or image files like JPG or PNG.	<code>"filepath": "calib_data/"</code>
<code>colorformat</code>	Color format that model expects. This will be used while converting images to binary. Supported values: [RGB, BGR]. Default is RGB.	<code>"colorformat":"RGB"</code>
<code>mean</code>	Mean value to be subtracted from the input. Can be a single value or a list of values. When the mean is given as a list the number of entries must match the channel dimension of the input.	<code>"mean":128.0</code>

Key	Description	Example
scale	Scale value to be used for normalizing the input. Can be a single value or a list of values. When the scale is given as a list, the number of entries must match the channel dimension of the input.	"scale": 255.0

The following is a sample configuration file:

```
{
    "inputs": {
        "data": {
            "shape": "1, 3, 224, 224",
            "filepath": "calib_data/",
            "colorformat": "RGB",
            "mean": [128,128,128],
            "scale": [128.0,128.0,128.0]
        }
    }
}
```

## Calibration Images

Quantize your trained model by providing calibration images. Quantizing your model improves the performance of the CVFlow engine on an Ambarella System on a Chip (SoC). The Ambarella toolchain uses the calibration images to determine how each layer in the model should be quantized to achieve optimal performance and accuracy. Each layer is quantized independently to INT8 or INT16 formats. The final model has a mix of INT8 and INT16 layers after quantization.

### How many images should you use?

It is recommended that you include between 100–200 images that are representative of the types of scenes the model is expected to handle. The model compilation time increases linearly to the number of calibration images in the input file.

### What are the recommended image formats?

Calibration images can be in a raw binary format or image formats such as JPG and PNG.

Your calibration folder can contain a mixture of images and binary files. If the calibration folder contains both images and binary files, the toolchain first converts the images to binary files. Once the conversion is complete, it uses the newly generated binary files along with the binary files that were originally in the folder.

### Can I convert the images into binary format first?

Yes. You can convert the images to the binary format with open-source packages such as [OpenCV](#) or [PIL](#). Crop and resize the images so they satisfy the input layer of your trained model.

## Mean and Scale

You can specify mean and scaling pre-processing options to the Amberalla toolchain. These operations are embedded into the network and are applied during inference on each input. Do not provide processed data if you specify the mean or scale. More specifically, do not provide data you have subtracted the mean from or have applied scaling to.

# SageMaker Edge Manager

Amazon SageMaker Edge Manager provides model management for edge devices so you can optimize, secure, monitor, and maintain machine learning models on fleets of edge devices such as smart cameras, robots, personal computers, and mobile devices.

## Why Use Edge Manager?

Many machine learning (ML) use cases require running ML models on a fleet of edge devices, which allows you to get predictions in real-time, preserves the privacy of the end users, and lowers the cost of network connectivity. With the increasing availability of low-power edge hardware designed for ML, it is now possible to run multiple complex neural network models on edge devices.

However, operating ML models on edge devices is challenging, because devices, unlike cloud instances, have limited compute, memory, and connectivity. After the model is deployed, you need to continuously monitor the models, because model drift can cause the quality of model to decay overtime. Monitoring models across your device fleets is difficult because you need to write custom code to collect data samples from your device and recognize skew in predictions. In addition, models are often hard-coded into the application. To update the model, you must rebuild and update the entire application or device firmware, which can disrupt your operations.

With SageMaker Edge Manager, you can optimize, run, monitor, and update machine learning models across fleets of devices at the edge.

## How Does it Work?

At a high level, there are five main components in the SageMaker Edge Manager workflow: compiling models with SageMaker Neo, packaging Neo-compiled models, deploying models to your devices, running models on the SageMaker inference engine (Edge Manager agent), and maintaining models on the devices.



SageMaker Edge Manager uses SageMaker Neo to optimize your models for the target hardware in one click, then to cryptographically sign your models before deployment. Using SageMaker Edge Manager, you can sample model input and output data from edge devices and send it to the cloud for monitoring and analysis, and view a dashboard that tracks and visually reports on the operation of the deployed models within the SageMaker console.

SageMaker Edge Manager extends capabilities that were previously only available in the cloud to the edge, so developers can continuously improve model quality by using Amazon SageMaker Model Monitor for drift detection, then relabel the data with SageMaker Ground Truth and retrain the models in SageMaker.

## How Do I Use SageMaker Edge Manager?

If you are a first time user of SageMaker Edge Manager, we recommend that you do the following:

1. **Read the [Getting Started](#) section** - This section walks you through setting up your first edge packaging job and creating your first fleet.
2. **Explore [Edge Manager Jupyter notebook examples](#)** - Example notebooks are stored in the [amazon-sagemaker-examples](#) GitHub repository in the [sagemaker\\_edge\\_manager](#) folder.

## Getting Started

This guide demonstrates how to complete the necessary steps to register, deploy, and manage a fleet of devices, and how to satisfy Amazon SageMaker Edge Manager prerequisites.

### Topics

- [Setting Up \(p. 2087\)](#)
- [Train, Compile, and Package Your Model \(p. 2090\)](#)
- [Create and Register Fleets and Authenticate Devices \(p. 2092\)](#)
- [Download and Set Up Edge Manager \(p. 2095\)](#)
- [Run Agent \(p. 2098\)](#)

## Setting Up

Before you begin using SageMaker Edge Manager to manage models on your device fleets, you must first create IAM Roles for both SageMaker and Amazon IoT. You will also want to create at least one Amazon S3 bucket where you will store your pre-trained model, the output of your SageMaker Neo compilation job, as well as input data from your edge devices.

### 1. Set up an Amazon account.

Create an Amazon account and an IAM administrator user. For instructions on how to set up your Amazon account, see [How do I create and activate a new Amazon account?](#) For instructions on how to create an administrator user in your Amazon account, see [Creating your first IAM admin user and group.](#)

### 2. Create an IAM role for Amazon SageMaker.

SageMaker Edge Manager needs access to your Amazon S3 bucket URI. To facilitate this, create an IAM role that can run SageMaker and has permission to access Amazon S3. Using this role, SageMaker can run under your account and access to your Amazon S3 bucket.

You can create an IAM role by using the IAM console, Amazon SDK for Python (Boto3), or Amazon CLI. The following is an example of how to create an IAM role and attach the necessary policies with the IAM console.

- a. Sign in to the Amazon Web Services Management Console and open the IAM console at <https://console.amazonaws.cn/iam/>.
- b. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
- c. For **Select type of trusted entity**, choose **Amazon service**.

- d. Choose the service that you want to allow to assume this role. In this case, choose **SageMaker**. Then choose **Next: Permissions**.
  - This automatically creates an IAM policy that grants access to related services such as Amazon S3, Amazon ECR, and CloudWatch Logs.
- e. Choose **Next: Tags**.
- f. (Optional) Add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see [Tagging IAM resources](#).
- g. Choose **Next: Review**.
- h. Type in a **Role name**.
- i. If possible, type a role name or role name suffix. Role names must be unique within your Amazon account. They are not distinguished by case. For example, you cannot create roles named both PRODROLE and prodrole. Because other Amazon resources might reference the role, you cannot edit the name of the role after it has been created.
- j. (Optional) For **Role description**, type a description for the new role.
- k. Review the role and then choose **Create role**.

Note the SageMaker Role ARN, which you use to create a compilation job with SageMaker Neo and a packaging job with Edge Manager. To find out the role ARN using the console, do the following:

- i. Go to the IAM console: <https://console.amazonaws.cn/iam/>
- ii. Select **Roles**.
- iii. Search for the role you just created by typing in the name of the role in the search field.
- iv. Select the role.
- v. The role ARN is at the top of the **Summary** page.

### 3. Create an IAM role for Amazon IoT.

The Amazon IoT IAM role you create is used to authorize your thing objects. You also use the IAM role ARN to create and register device fleets with a SageMaker client object.

Configure an IAM role in your Amazon account for the credentials provider to assume on behalf of the devices in your device fleet. Then, attach a policy to authorize your devices to interact with Amazon IoT services.

Create a role for Amazon IoT either programmatically or with the IAM console, similar to what you did when you created a role for SageMaker.

- a. Sign in to the Amazon Web Services Management Console and open the IAM console at <https://console.amazonaws.cn/iam/>.
- b. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
- c. For **Select type of trusted entity**, choose **Amazon service**.
- d. Choose the service that you want to allow to assume this role. In this case, choose **IoT**. Select **IoT** as the **Use Case**.
- e. Choose **Next: Permissions**.
- f. Choose **Next: Tags**.
- g. (Optional) Add metadata to the role by attaching tags as key–value pairs. For more information about using tags in IAM, see [Tagging IAM resources](#).
- h. Choose **Next: Review**.
- i. Type in a **Role name**. The role name must start with **SageMaker**.
- j. (Optional) For **Role description**, type a description for the new role.
- k. Review the role and then choose **Create role**.

- l. Once the role is created, choose **Roles** in the IAM console. Search for the role you created by typing in role name in the **Search** field.
- m. Choose your role.
- n. Next, choose **Attach Policies**.
- o. Search for `AmazonSageMakerEdgeDeviceFleetPolicy` in the **Search** field. Select `AmazonSageMakerEdgeDeviceFleetPolicy`.
- p. Choose **Attach policy**.
- q. Add the following policy statement to the trust relationship:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {"Service": "credentials.iot.amazonaws.com"},  
            "Action": "sts:AssumeRole"  
        },  
        {  
            "Effect": "Allow",  
            "Principal": {"Service": "sagemaker.amazonaws.com"},  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

A trust policy is a [JSON policy document](#) in which you define the principals that you trust to assume the role. For more information about trust policies, see [Roles terms and concepts](#).

- r. Note the Amazon IoT role ARN. You use the Amazon IoT Role ARN to create and register the device fleet. To find the IAM role ARN with the console:
  - i. Go to the IAM console: <https://console.amazonaws.cn/iam/>
  - ii. Choose **Roles**.
  - iii. Search for the role you created by typing in the name of the role in the **Search** field.
  - iv. Select the role.
  - v. The role ARN is on the Summary page.

#### 4. Create an Amazon S3 bucket.

SageMaker Neo and Edge Manager access your pre-compiled model and compiled model from an Amazon S3 bucket. Edge Manager also stores sample data from your device fleet in Amazon S3.

- a. Open the Amazon S3 console at <https://console.amazonaws.cn/s3/>.
- b. Choose **Create bucket**.
- c. In **Bucket name**, enter a name for your bucket.
- d. In **Region**, choose the Amazon Region where you want the bucket to reside.
- e. In **Bucket settings for Block Public Access**, choose the settings that you want to apply to the bucket.
- f. Choose **Create bucket**.

For more information about creating Amazon S3 buckets, see [Getting started with Amazon S3](#).

Store the name of the bucket in the following variable:

```
bucket=<bucket-name>
```

## Train, Compile, and Package Your Model

In this section you will create SageMaker and Amazon IoT client objects, download a pre-trained machine learning model, upload your model to your Amazon S3 bucket, compile your model for your target device with SageMaker Neo, and package your model so that it can be deployed with the Edge Manager agent.

### 1. Import libraries and create client objects.

This tutorial uses the Amazon SDK for Python (Boto3) to create clients to interact with SageMaker, Amazon S3, and Amazon IoT.

Import Boto3, specify your Region, and initialize the client objects you need as shown in the following example:

```
import boto3
import json
import time

AWS_REGION = 'us-west-2'# Specify your Region
bucket = 'edge-manager-demo-bucket'

sagemaker_client = boto3.client('sagemaker', region_name=AWS_REGION)
iot_client = boto3.client('iot', region_name=AWS_REGION)
```

Define variables and assign them the role ARN you created for SageMaker and Amazon IoT as strings:

```
# Replace with the role ARN you created for SageMaker
sagemaker_role_arn = "arn:aws:iam::<account>:role/*"

# Replace with the role ARN you created for Amazon IoT.
# Note: The name must start with 'SageMaker'
iot_role_arn = "arn:aws:iam::<account>:role/SageMaker*"
```

### 2. Train a machine learning model.

See [Train a Model with Amazon SageMaker](#) for more information on how to train a machine learning model using SageMaker. You can optionally upload your locally trained model directly into an Amazon S3 URI bucket.

If you do not have a model yet, use the curl command to get a local copy of the coco\_ssd\_mobilenet model from TensorFlow's website. The model you just copied is an object detection model trained from the [COCO dataset](#). Type the following into your Jupyter Notebook:

```
model_zip_filename = './coco_ssd_mobilenet_v1_1.0.zip'
!curl http://storage.googleapis.com/download.tensorflow.org/models/tflite/
coco_ssd_mobilenet_v1_1.0_quant_2018_06_29.zip \ --output {model_zip_filename}
```

This particular example was packaged in a .zip file. Unzip this file and repackage it as a compressed TAR file (.tar.gz) before using it in later steps. Type the following into your Jupyter Notebook:

```
# Extract model from zip file
!unzip -u {model_zip_filename}

model_filename = 'detect.tflite'
model_name = model_filename.split('.')[0]

# Compress model into .tar.gz so SageMaker Neo can use it
```

```
model_filename = model_name + '.tar.gz'  
!tar -czf {model_tar} {model_filename}
```

### 3. Upload your model to Amazon S3.

Once you have a machine learning model, store it in an Amazon S3 bucket. The following example uses an Amazon CLI command to upload the model to the Amazon S3 bucket you created earlier in a directory called *models*. Type in the following into your Jupyter Notebook:

```
!aws cp detect.tar.gz s3://{bucket}/models/
```

### 4. Compile your model with SageMaker Neo.

Compile your machine learning model with SageMaker Neo for an edge device. You need to know your Amazon S3 bucket URI where you stored the trained model, the machine learning framework you used to train your model, the shape of your model's input, and your target device.

For the Keras MobileNet V2 model, use the following:

```
framework = 'tflite'  
target_device = 'rasp3b' # Raspberry Pi 3  
data_shape = '{"normalized_input_image_tensor": [1, 300, 300, 3]}'
```

SageMaker Neo requires a specific model input shape and model format based on the deep learning framework you use. For more information about how to save your model, see [What input data shapes does SageMaker Neo expect? \(p. 2023\)](#). For more information about devices and frameworks supported by Neo, see [Supported Frameworks, Devices, Systems, and Architectures \(p. 2063\)](#).

Use the `CreateCompilationJob` API to create a compilation job with SageMaker Neo. Provide a name to the compilation job, the SageMaker Role ARN, the Amazon S3 URI where your model is stored, the input shape of the model, the name of the framework, the Amazon S3 URI where you want SageMaker to store your compiled model, and your edge device target.

```
# Specify the path where your model is stored  
model_directory = 'models'  
s3_model_uri = 's3://{}//{}//{}'.format(bucket, model_directory, model_filename)  
  
# Store compiled model in S3 within the 'compiled-models' directory  
compilation_output_dir = 'compiled-models'  
s3_output_location = 's3://{}//{}//{}'.format(bucket, compilation_output_dir)  
  
# Give your compilation job a name  
compilation_job_name = 'getting-started-demo'  
  
sagemaker_client.create_compilation_job(CompilationJobName=compilation_job_name,  
                                         RoleArn=sagemaker_role_arn,  
                                         InputConfig={  
                                             'S3Uri': s3_model_uri,  
                                             'DataInputConfig': data_shape,  
                                             'Framework': framework.upper()},  
                                         OutputConfig={  
                                             'S3OutputLocation': s3_output_location,  
                                             'TargetDevice': target_device},  
                                         StoppingCondition={'MaxRuntimeInSeconds': 900})
```

### 5. Package your compiled model.

Packaging jobs take SageMaker Neo-compiled models and make any changes necessary to deploy the model with the inference engine, Edge Manager agent. To package your model, create an edge packaging job with the `create_edge_packaging` API or the SageMaker console.

You need to provide the name that you used for your Neo compilation job, a name for the packaging job, a role ARN (see [Setting Up \(p. 2087\)](#) section), a name for the model, a model version, and the Amazon S3 bucket URI for the output of the packaging job. Note that Edge Manager packaging job names are case-sensitive. The following is an example of how to create a packaging job using the API.

```
edge_packaging_name='edge-packaging-demo'  
model_name="sample-model"  
model_version="1.1"
```

Define the Amazon S3 URI where you want to store the packaged model.

```
# Output directory where you want to store the output of the packaging job  
packaging_output_dir = 'packaged_models'  
packaging_s3_output = 's3://{}{}'.format(bucket, packaging_output_dir)
```

Use `CreateEdgePackagingJob` to package your Neo-compiled model. Provide a name for your edge packaging job and the name you provided for your compilation job (in this example, it was stored in the variable `compilation_job_name`). Also provide a name for your model, a version for your model (this is used to help you keep track of what model version you are using), and the S3 URI where you want SageMaker to store the packaged model.

```
sagemaker_client.create_edge_packaging_job(  
    EdgePackagingJobName=edge_packaging_name,  
    CompilationJobName=compilation_job_name,  
    RoleArn=sagemaker_role_arn,  
    ModelName=model_name,  
    ModelVersion=model_version,  
    OutputConfig={  
        "S3OutputLocation": packaging_s3_output  
    }  
)
```

## Create and Register Fleets and Authenticate Devices

In this section you will create your Amazon IoT thing object, create a device fleet, register your device fleet so it can interact with the cloud, create X.509 certificates to authenticate your devices to Amazon IoT Core, associate the role alias with Amazon IoT that was generated when you created your fleet, get your Amazon account-specific endpoint for the credentials provider, get an official Amazon Root CA file, and upload the Amazon CA file to Amazon S3.

### 1. Create Amazon IoT things.

SageMaker Edge Manager takes advantage of the Amazon IoT Core services to facilitate the connection between the edge devices and endpoints in the Amazon cloud. You can take advantage of existing Amazon IoT functionality after you set up your devices to work with Edge Manager.

To connect your device to Amazon IoT, you need to create Amazon IoT *thing objects*, create and register a client certificate with Amazon IoT, and create and configure the IAM role for your devices.

First, create Amazon IoT thing objects with the Amazon IoT client (`iot_client`) you created earlier with Boto3. The following example shows how to create two thing objects:

```

iot_thing_name = 'sample-device'
iot_thing_type = 'getting-started-demo'

iot_client.create_thing_type(
    thingTypeName=iot_thing_type
)

# Create an Amazon IoT thing objects
iot_client.create_thing(
    thingName=iot_thing_name,
    thingTypeName=iot_thing_type
)

```

## 2. Create your device fleet.

Create a device fleet with the SageMaker client object defined in a previous step. You can also use the SageMaker console to create a device fleet.

```

device_fleet_name="demo-device-fleet" + str(time.time()).split('.')[0]
device_name="sagemaker-edge-demo-device" + str(time.time()).split('.')[0]

```

Specify your IoT role ARN. This lets Amazon IoT grant temporary credentials to devices.

```

device_model_directory='device_output'
s3_device_fleet_output = 's3://{}{}'.format(bucket, device_model_directory)

sagemaker_client.create_device_fleet(
    DeviceFleetName=device_fleet_name,
    RoleArn=iot_role_arn, # IoT Role ARN specified in previous step
    OutputConfig={
        'S3OutputLocation': s3_device_fleet_output
    }
)

```

An Amazon IoT role alias is created when you create a device fleet. This role alias is associated with Amazon IoT using the `iot_client` object in a later step.

## 3. Register your device fleet.

To interact with the cloud, you need to register your device with SageMaker Edge Manager. In this example, you register a single device with the fleet you created. To register the device, you need to provide a device name and the Amazon IoT thing name as shown in the following example:

```

# Device name should be 36 characters
device_name = "sagemaker-edge-demo-device" + str(time.time()).split('.')[0]

sagemaker_client.register_devices(
    DeviceFleetName=device_fleet_name,
    Devices=[
        {
            "DeviceName": device_name,
            "IotThingName": iot_thing_name
        }
    ]
)

```

## 4. Create X.509 certificates.

After creating the Amazon IoT thing object, you must create a X.509 device certificate for your thing object. This certificate authenticates your device to Amazon IoT Core.

Use the following to create a private key, public key, and a X.509 certificate file using the Amazon IoT client defined (`iot_client`) earlier.

```
# Creates a 2048-bit RSA key pair and issues an X.509 certificate
# using the issued public key.
create_cert = iot_client.create_keys_and_certificate(
    setAsActive=True
)

# Get certificate from dictionary object and save in its own
with open('./device.pem.crt', 'w') as f:
    for line in create_cert['certificatePem'].split('\n'):
        f.write(line)
        f.write('\n')

# Get private key from dictionary object and save in its own
with open('./private.pem.key', 'w') as f:
    for line in create_cert['keyPair']['PrivateKey'].split('\n'):
        f.write(line)
        f.write('\n')

# Get a private key from dictioanary object and save in its own
with open('./public.pem.key', 'w') as f:
    for line in create_cert['keyPair']['PublicKey'].split('\n'):
        f.write(line)
        f.write('\n')
```

## 5. Associate the role alias with Amazon IoT.

When you create a device fleet with SageMaker (`sagemaker_client.create_device_fleet()`), a role alias is generated for you. An Amazon IoT role alias provides a mechanism for connected devices to authenticate to Amazon IoT using X.509 certificates, and then obtain short-lived Amazon credentials from an IAM role that is associated with an Amazon IoT role alias. The role alias allows you to change the role of the device without having to update the device. Use `DescribeDeviceFleet` to get the role alias name and ARN.

```
# Print Amazon Resource Name (ARN) and alias that has access
# to Amazon Internet of Things (IoT).
sagemaker_client.describe_device_fleet(DeviceFleetName=device_fleet_name)

# Store iot role alias string in a variable
# Grabs role ARN
full_role_alias_name =
    sagemaker_client.describe_device_fleet(DeviceFleetName=device_fleet_name)
['IotRoleAlias']
start_index = full_role_alias_name.find('SageMaker') # Find beginning of role name
role_alias_name = full_role_alias_name[start_index:]
```

Use the `iot_client` to facilitate associating the role alias generated from creating the device fleet with Amazon IoT:

```
role_alias = iot_client.describe_role_alias(
    roleAlias=role_alias_name)
```

For more information about IAM role alias, see [Role alias allows access to unused services](#).

You created and registered a certificate with Amazon IoT earlier for successful authentication of your device. Now, you need to create and attach a policy to the certificate to authorize the request for the security token.

```
alias_policy = {
```

```

    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "iot:AssumeRoleWithCertificate",
            "Resource": role_alias['roleAliasDescription']['roleAliasArn']
        }
    }

policy_name = 'aliaspolicy-' + str(time.time()).split('.')[0]
aliaspolicy = iot_client.create_policy(policyName=policy_name,
                                        policyDocument=json.dumps(alias_policy))

# Attach policy
iot_client.attach_policy(policyName=policy_name,
                         target=create_cert['certificateArn'])

```

## 6. Get your Amazon account-specific endpoint for the credentials provider.

Edge devices need an endpoint in order to assume credentials. Obtain your Amazon account-specific endpoint for the credentials provider.

```

# Get the unique endpoint specific to your Amazon account that is making the call.
iot_endpoint = iot_client.describe_endpoint(
    endpointType='iot:CredentialProvider'
)

endpoint="https://{}//role-aliases/{}/credentials".format(iot_endpoint['endpointAddress'],role_alias_name)

```

## 7. Get the official Amazon root CA file and upload it to the Amazon S3 bucket.

Use the following in your Jupyter Notebook or Amazon CLI (if you use your terminal, remove the '!' magic function):

```
!wget https://www.amazontrust.com/repository/AmazonRootCA1.pem
```

Use the endpoint to make an HTTPS request to the credentials provider to return a security token. The following example command uses curl, but you can use any HTTP client.

```
!curl --cert device.pem.crt --key private.pem.key --cacert AmazonRootCA1.pem $endpoint
```

If the certificate is verified, upload the keys and certificate to your Amazon S3 bucket URI:

```
!aws s3 cp private.pem.key s3://{}bucket}/authorization-files/
!aws s3 cp device.pem.crt s3://{}bucket}/authorization-files/
!aws s3 cp AmazonRootCA1.pem s3://{}bucket}/authorization-files/
```

## Download and Set Up Edge Manager

The Edge Manager agent is an inference engine for your edge devices. Use the agent to make predictions with models loaded onto your edge devices. The agent also collects model metrics and captures data at specific intervals.

In this section you will set up your device with the agent. To do so, first copy a release artifact and signing root certificate from the release bucket locally to your machine. After you unzip the release artifact, upload it to Amazon S3. Next, define and save a configuration file for the agent. A template is

provided for you to copy and paste. Finally, copy the release artifacts, configuration file, and credentials to your device.

### 1. Download the SageMaker Edge Manager agent.

The agent is released in binary format for supported operating systems. This example runs inference on a Linux platform (Raspberry Pi OS is based on Debian). Fetch the latest version of binaries from the SageMaker Edge Manager release bucket from the us-west-2 Region.

```
aws s3 ls s3://sagemaker-edge-release-store-us-west-2-windows-x86 Releases/ | sort -r
```

This returns release artifacts sorted by their version.

```
2020-12-01 23:33:36 0  
PRE 1.20201218.81f481f/  
PRE 1.20201207.02d0e97/
```

The version has the following format: <MAJOR\_VERSION>. <YYYY-MM-DD>-<SHA-7>. It consists of three components:

- <MAJOR\_VERSION>: The release version. The release version is currently set to 1.
- <YYYY-MM-DD>: The time stamp of the artifact release.
- <SHA-7>: The repository commit ID from which the release is built.

Copy the zipped TAR file locally or to your device directly. The following example shows how to copy the latest release artifact at the time this document was released.

```
!aws s3 cp s3://sagemaker-edge-release-store-us-west-2-linux-x64/  
Releases/1.20201218.81f481f/1.20201218.81f481f.tgz ./
```

Once you have the artifact, unzip the zipped TAR file:

```
!mkdir agent_demo  
!tar -xvzf 1.20201218.81f481f.tgz -C ./agent_demo
```

You also need the signing root certificates from the release bucket:

```
!aws s3 cp s3://sagemaker-edge-release-store-us-west-2-linux-x64/Certificates/us-  
west-2/us-west-2.pem ./
```

### 2. Define a SageMaker Edge Manager agent configuration file.

First, define the agent configuration file as follows:

```
sagemaker_edge_config = {  
    "sagemaker_edge_core_device_name": "device_name",  
    "sagemaker_edge_core_device_fleet_name": "device_fleet_name",  
    "sagemaker_edge_core_capture_data_buffer_size": 30,  
    "sagemaker_edge_core_capture_data_push_period_seconds": 4,  
    "sagemaker_edge_core_folder_prefix": "demo_capture",  
    "sagemaker_edge_core_region": "us-west-2",  
    "sagemaker_edge_core_root_certs_path": "/agent_demo/certificates",  
    "sagemaker_edge_provider_aws_ca_cert_file": "/agent_demo/iot-credentials/  
AmazonRootCA1.pem",
```

```
        "sagemaker_edge_provider_aws_cert_file": "/agent_demo/iot-credentials/  
device.pem.crt",  
        "sagemaker_edge_provider_aws_cert_pk_file": "/agent_demo/iot-credentials/  
private.pem.key",  
        "sagemaker_edge_provider_aws_iot_cred_endpoint": "endpoint",  
        "sagemaker_edge_provider_provider": "Aws",  
        "sagemaker_edge_provider_s3_bucket_name": bucket,  
        "sagemaker_edge_core_capture_data_destination": "Cloud"  
    }
```

Next, save it as a JSON file:

```
edge_config_file = open("sagemaker_edge_config.json", "w")  
json.dump(sagemaker_edge_config, edge_config_file, indent = 6)  
edge_config_file.close()
```

Replace the following:

- "device\_name" with the name of your device (this string was stored in an earlier step in a variable named device\_name).
- "device\_fleet\_name" with the name of your device fleet (this string was stored an earlier step in a variable named device\_fleet\_name)
- "endpoint" with your Amazon account-specific endpoint for the credentials provider (this string was stored in an earlier step in a variable named endpoint).

### 3. Copy the release artifacts, configuration file, and credentials to your device.

The following instructions are performed on the edge device itself.

**Note**

You must first install Python, the Amazon SDK for Python (Boto3), and the Amazon CLI on your edge device.

Open a terminal on your device. Create a folder to store the release artifacts, your credentials, and the configuration file.

```
mkdir agent_demo  
cd agent_demo
```

Copy the contents of the release artifacts that you stored in your Amazon S3 bucket to your device:

```
# Copy release artifacts  
aws s3 cp s3://sagemaker-edge-manager-demo/release_artifacts/ ./ --recursive
```

Go the /bin directory and make the binary files executable:

```
cd bin  
  
chmod +x sagemaker_edge_agent_binary  
chmod +x sagemaker_edge_agent_client_example  
  
cd agent_demo
```

Make a directory to store your Amazon IoT credentials and copy your credentials from your Amazon S3 bucket to your edge device (use the same on you define in the variable bucket):

```
mkdir iot-credentials  
cd iot-credentials
```

```
aws s3 cp s3://<bucket-name>/authorization-files/AmazonRootCA1.pem ./
aws s3 cp s3://<bucket-name>/authorization-files/device.pem.crt ./
aws s3 cp s3://<bucket-name>/authorization-files/private.pem.key ./

cd ../
```

Make a directory to store your model signing root certificates:

```
mkdir certificates
cd certificates
aws s3 cp s3://<bucket-name>/certificates/us-west-2.pem ./
cd agent_demo
```

Copy your configuration file to your device:

```
#Download config file from S3
aws s3 cp s3://<bucket-name>/config_files/sagemaker_edge_config.json ./
cd agent_demo
```

Your agent\_demo directory on your edge device should look similar to the following:

```
###agent_demo
|   ### bin
|       ### sagemaker_edge_agent_binary
|       ### sagemaker_edge_agent_client_example
|   ### sagemaker_edge_config.json
|   ### certificates
|       ###us-west-2.pem
|   ### iot-credentials
|       ### AmazonRootCA1.pem
|       ### device.pem.crt
|       ### private.pem.key
|   ### docs
|       ### api
|       ### examples
|   ### ATTRIBUTIONS.txt
|   ### LICENSE.txt
|   ### RELEASE_NOTES.md
```

## Run Agent

In this section you will run the agent as a binary using gRPC, and check that both your device and fleet are working and collecting sample data.

### 1. Launch the agent.

The SageMaker Edge Manager agent can be run as a standalone process in the form of an Executable and Linkable Format (ELF) executable binary or can be linked against as a Dynamic Shared Object (.dll). Running as a standalone executable binary is the preferred mode and is supported on Linux.

This example uses gRPC to run the agent. gRPC is an open source high-performance Remote Procedure Call (RPC) framework that can run in any environment. For more information about gRPC, see the [gRPC documentation](#).

To use gRPC, perform the following steps:

- a. Define a service in a .proto file.
- b. Generate server and client code using the protocol buffer compiler.
- c. Use the Python (or other languages supported by gRPC) gRPC API to write the server for your service.
- d. Use the Python (or other languages supported by gRPC) gRPC API to write a client for your service.

The release artifact you downloaded contains a gRPC application ready for you to run the agent. The example is located within the /bin directory of your release artifact. The sagemaker\_edge\_agent\_binary executable is in this directory.

To run the agent with this example, provide the path to your socket file (.sock) and JSON .config file:

```
./bin/sagemaker_edge_agent_binary -a /tmp/sagemaker_edge_agent_example.sock -c sagemaker_edge_config.json
```

## 2. Check your device.

Check that your device is connected and sampling data. Making periodic checks, manually or automatically, allows you to check that your device or fleet is working properly.

Provide the name of the fleet to which the device belongs and the unique device identifier. From your local machine, run the following:

```
sagemaker_client.describe_device(  
    DeviceName=device_name,  
    DeviceFleetName=device_fleet_name  
)
```

For the given model, you can see the name, model version, latest sample time, and when the last inference was made.

```
{  
    "DeviceName": "sample-device",  
    "DeviceFleetName": "demo-device-fleet",  
    "IoTThingName": "sample-thing-name-1",  
    "RegistrationTime": 1600977370,  
    "LatestHeartbeat": 1600977370,  
    "Models": [  
        {  
            "ModelName": "mobilenet_v2.tar.gz",  
            "ModelVersion": "1.1",  
            "LatestSampleTime": 1600977370,  
            "LatestInference": 1600977370  
        }  
    ]  
}
```

The timestamp provided by `LastetHeartbeat` indicates the last signal that was received from the device. `LatestSampleTime` and `LatestInference` describe the time stamp of the last data sample and inference, respectively.

## 3. Check your fleet.

Check that your fleet is working with `GetDeviceFleetReport`. Provide the name of the fleet the device belongs to.

```
sagemaker_client.get_device_fleet_report(  
    DeviceFleetName=device_fleet_name  
)
```

For a given model, you can see the name, model version, latest sample time, and when the last inference was made, along with the Amazon S3 bucket URL where the data samples are stored.

```
# Sample output  
{  
    "DeviceFleetName": "sample-device-fleet",  
    "DeviceFleetArn": "arn:aws:sagemaker:us-west-2:999999999999:device-fleet/sample-fleet-name",  
    "OutputConfig": {  
        "S3OutputLocation": "s3://fleet-bucket/package_output",  
    },  
    "AgentVersions": [{"Version": "1.1", "AgentCount": 2}],  
    "DeviceStats": {"Connected": 2, "Registered": 2},  
    "Models": [  
        {"ModelName": "sample-model",  
         "ModelVersion": "1.1",  
         "OfflineDeviceCount": 0,  
         "ConnectedDeviceCount": 2,  
         "ActiveDeviceCount": 2,  
         "SamplingDeviceCount": 100  
    ]  
}
```

## Set Up Devices and Fleets

Fleets are collections of logically grouped devices you can use to collect and analyze data. You can use SageMaker Edge Manager to operate machine learning models on a fleet of smart cameras, smart speakers, robots, and other edge devices.

Create a fleet and register your devices either programmatically with the Amazon SDK for Python (Boto3) or through the SageMaker console.

### Topics

- [Create a Fleet \(p. 2100\)](#)
- [Register a Device \(p. 2104\)](#)
- [Check Status \(p. 2106\)](#)

## Create a Fleet

You can create a fleet programmatically with the Amazon SDK for Python (Boto3) or through the SageMaker console <https://console.amazonaws.cn/sagemaker>.

### Create a Fleet (Boto3)

Use the `CreateDeviceFleet` API to create a fleet. Specify a name for the fleet, your Amazon IoT Role ARN for the `RoleArn` field, as well as an Amazon S3 URI where you want the device to store sampled data.

You can optionally include a description of the fleet, tags, and an Amazon KMS Key ID.

```
import boto3

# Create SageMaker client so you can interact and manage SageMaker resources
sagemaker_client = boto3.client("sagemaker", region_name="aws-region")

sagemaker_client.create_device_fleet(
    DeviceFleetName="sample-fleet-name",
    RoleArn="arn:aws:iam::9999999999:role/rolename", # IoT Role ARN
    Description="fleet description",
    OutputConfig={
        S3OutputLocation="s3://bucket/",
        KMSKeyId: "1234abcd-12ab-34cd-56ef-1234567890ab",
    },
    Tags=[
        {
            "Key": "string",
            "Value" : "string"
        }
    ],
)
```

An Amazon IoT Role Alias is created for you when you create a device fleet. The Amazon IoT role alias provides a mechanism for connected devices to authenticate to Amazon IoT using X.509 certificates and then obtain short-lived Amazon credentials from an IAM role that is associated with the Amazon IoT role alias.

Use `DescribeDeviceFleet` to get the role alias name and ARN.

```
# Print Amazon Resource Name (ARN) and alias that has access
# to Amazon Internet of Things (IoT).
sagemaker_client.describe_device_fleet(DeviceFleetName=device_fleet_name)['IoTRoleAlias']
```

Use `DescribeDeviceFleet` API to get a description of fleets you created.

```
sagemaker_client.describe_device_fleet(
    DeviceFleetName="sample-fleet-name"
)
```

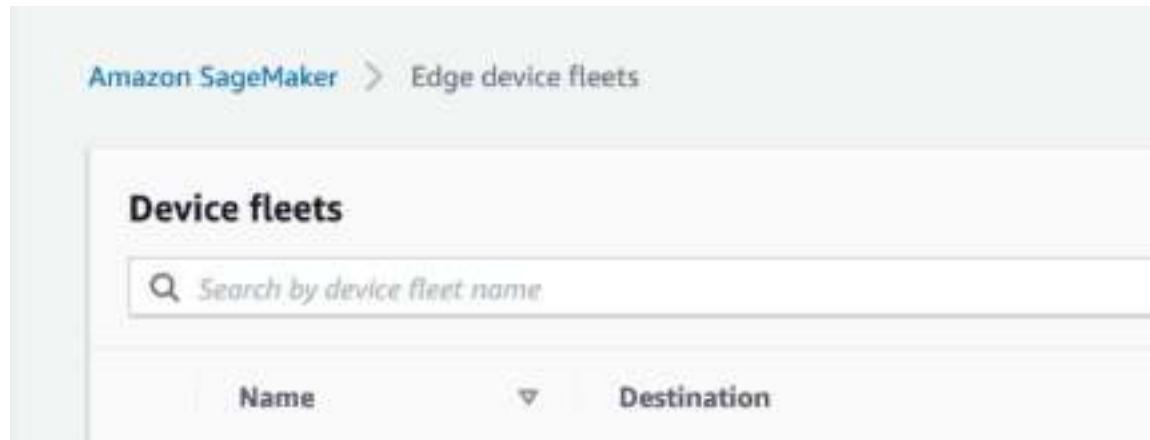
By default, it returns the name of the fleet, the device fleet ARN, the Amazon S3 bucket URI, the IAM role, the role alias created in Amazon IoT, a timestamp of when the fleet was created, and a timestamp of when the fleet was last modified.

```
{
    "DeviceFleetName": "sample-fleet-name",
    "DeviceFleetArn": "arn:aws:sagemaker:us-west-2:9999999999:device-fleet/sample-fleet-name",
    "IAMRole": "arn:aws:iam::9999999999:role/rolename",
    "Description": "this is a sample fleet",
    "IoTRoleAlias": "arn:aws:iot:us-west-2:9999999999:rolealias/SagemakerEdge-sample-fleet-name",
    "OutputConfig": {
        "S3OutputLocation": "s3://bucket/folder",
        "KMSKeyId": "1234abcd-12ab-34cd-56ef-1234567890ab"
    },
    "CreationTime": "1600977370",
    "LastModifiedTime": "1600977370"
}
```

## Create a Fleet (Console)

You can create a Edge Manager packaging job using the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker>.

1. In the SageMaker console, choose **Edge Inference** and then choose **Edge device fleets**.
2. Choose **Create device fleet**.



3. Enter a name for the device fleet in the **Device fleet name** field. Choose **Next**.

## Device fleet properties

Use the fields below to enter the name and the role for your device fleet.

Device fleet name

device-fleet-test

Device fleet description - *optional*

*Enter a description here*

512 character max

IAM role - *optional*

The role for  IoT to use when granting temporary

*Enter the role ARN*

Device fleet tags - *optional*

Key

4. On the **Output configuration** page, specify the Amazon S3 bucket URI where you want to store sample data from your device fleet. You can optionally add an encryption key as well by electing an existing Amazon KMS key from the dropdown list or by entering a key's ARN. Choose **Submit**.

The screenshot shows the 'Output configuration' section of a device fleet setup. It includes fields for an S3 bucket URI (containing 's3://bucket-example/sagemaker-edge/device-output'), an optional encryption key (set to 'No Custom Encryption'), and a 'Cancel' button.

**Output configuration**

Use the fields below to specify the S3 bucket URI where you want devices to store sample data. You can also (optionally) specify a KMS key.

**S3 bucket URI**  
Enter your S3 bucket URI where you want devices to store sample data.  
`s3://bucket-example/sagemaker-edge/device-output`

To find a path, [go to Amazon S3](#)

**Encryption key - optional**  
Encrypt your data. Choose an existing KMS key or enter a key's ARN.  
`No Custom Encryption`

[Cancel](#)

5. Choose the name of your device fleet to be redirected to the device fleet details. This page displays the name of the device fleet, ARN, description (if you provided one), date the fleet was created, last time the fleet was modified, Amazon S3 bucket URI, Amazon KMS key ID (if provided), Amazon IoT alias (if provided), and IAM role. If you added tags, they appear in the **Device fleet tags** section.

## Register a Device

### Important

Device registration is required to use any part of SageMaker Edge Manager.

You can create a fleet programmatically with the Amazon SDK for Python (Boto3) or through the SageMaker console at <https://console.amazonaws.cn/sagemaker>.

### Register a Device (Boto3)

To register your device, first create and register an Amazon IoT thing object and configure an IAM role. SageMaker Edge Manager takes advantage of the Amazon IoT Core services to facilitate the connection between the edge devices and the cloud. You can take advantage of existing Amazon IoT functionality after you set up your devices to work with Edge Manager.

To connect your device to Amazon IoT you need to create Amazon IoT thing objects, create and register a client certificate with Amazon IoT, and create and configure IAM role for your devices.

See the [Getting Started Guide](#) for an in-depth example or the [Explore Amazon IoT Core services in hands-on tutorial](#).

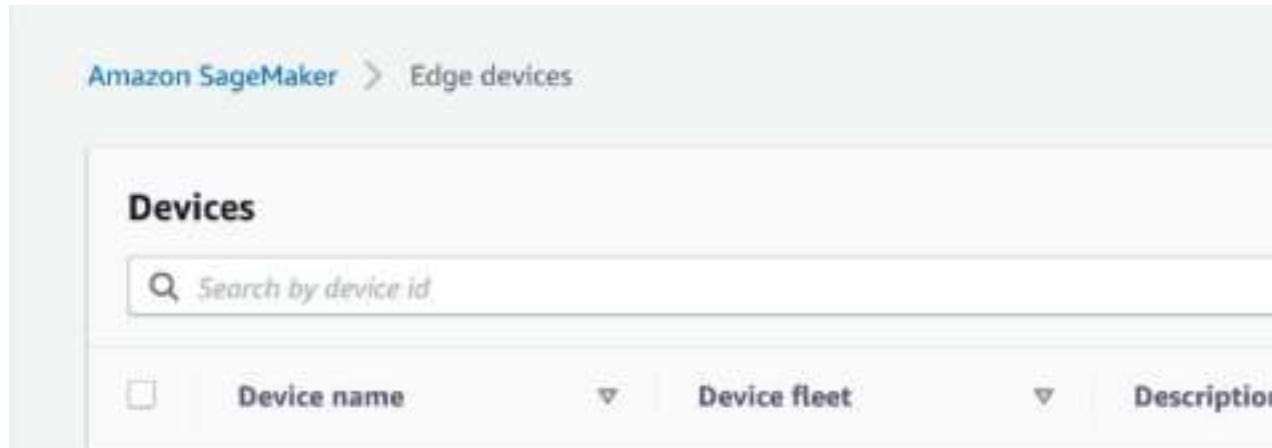
Use the `RegisterDevices` API to register your device. Provide the name of the fleet of which you want the devices to be a part, as well as a name for the device. You can optionally add a description to the device, tags, and Amazon IoT thing name associated with the device.

```
sagemaker_client.register_devices(  
    DeviceFleetName="sample-fleet-name",  
    Devices=[  
        {  
            "DeviceName": "sample-device-1",  
            "IoTThingName": "sample-thing-name-1",  
            "Description": "Device #1"  
        }  
    ],  
    Tags=[  
        {  
            "Key": "string",  
            "Value" : "string"  
        }  
    ]  
)
```

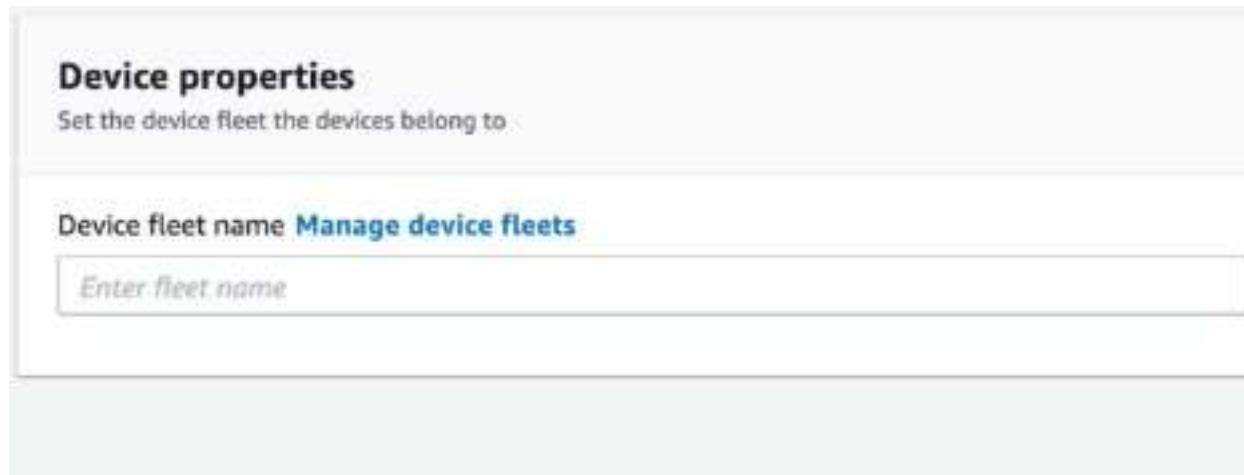
## Register a Device (Console)

You can register your device using the SageMaker console at <https://console.amazonaws.cn/sagemaker>.

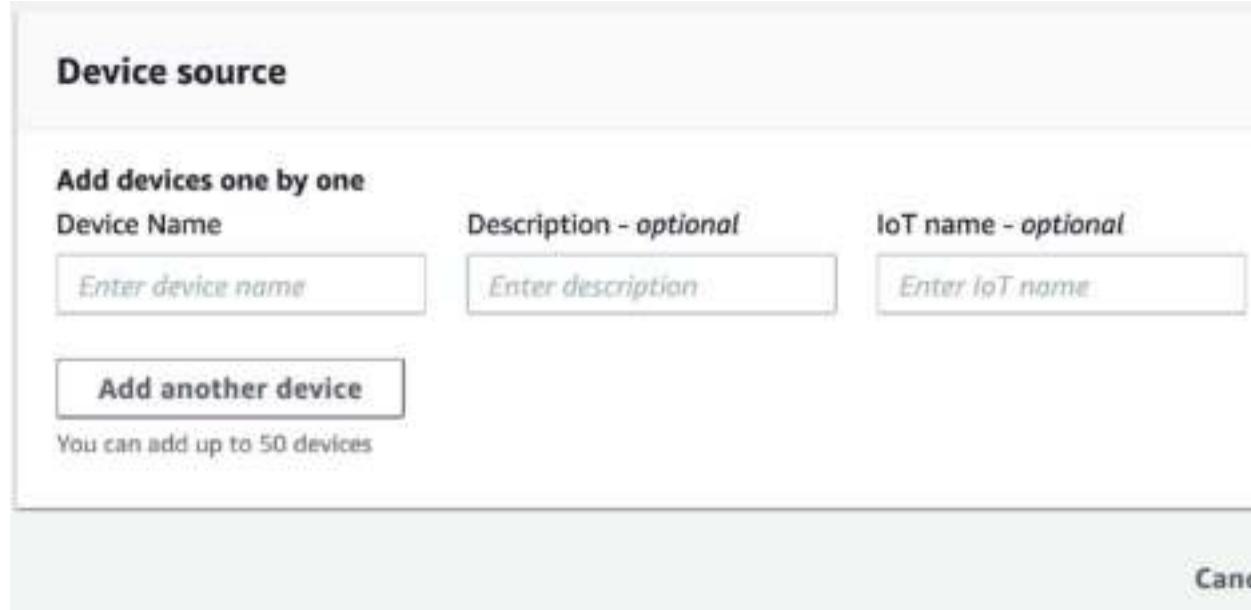
1. In the SageMaker console, choose **Edge Inference** and then choose **Edge devices**.
2. Choose **Register devices**.



3. In the **Device properties** section, enter the name of the fleet the device belongs to under the **Device fleet name** field. Choose **Next**.



4. In the **Device source** section, add your devices one by one. You must include a **Device Name** for each device in your fleet. You can optionally provide a description (in the **Description** field) and an Internet of Things (IoT) object name (in the **IoT name** field). Choose **Submit** once you have added all your devices.



The **Devices** page displays the name of the device you have added, the fleet to which it belongs, when it was registered, the last heartbeat, and the description and Amazon IoT name, if you provided one.

Choose a device to view the device's details, including the device name, fleet, ARN, description, IoT Thing name, when the device was registered, and the last heartbeat.

## Check Status

Check that your device or fleet is connected and sampling data. Making periodic checks, manually or automatically, allows you to check that your device or fleet is working properly.

Use the Amazon S3 console at <https://console.amazonaws.cn/s3/> to interactively choose a fleet for a status check. You can also use the Amazon SDK for Python (Boto3). The following describes different APIs from Boto3 you can use to check the status of your device or fleet. Use the API that best fits your use case.

- **Check an individual device.**

To check the status of an individual device, use `DescribeDevice` API. A list containing one or more models is provided if a models have been deployed to the device.

```
sagemaker_client.describe_device(  
    DeviceName="sample-device-1",  
    DeviceFleetName="sample-fleet-name"  
)
```

Running `DescribeDevice` returns:

```
{ "DeviceName": "sample-device".
```

```
"Description": "this is a sample device",
"DeviceFleetName": "sample-device-fleet",
"IoTThingName": "SampleThing",
"RegistrationTime": 1600977370,
"LatestHeartbeat": 1600977370,
"Models": [
    {
        "ModelName": "sample-model",
        "ModelVersion": "1.1",
        "LatestSampleTime": 1600977370,
        "LatestInference": 1600977370
    }
]
```

- **Check a fleet of devices.**

To check the status of the fleet, use the `GetDeviceFleetReport` API. Provide the name of the device fleet to get a summary of the fleet.

```
sagemaker_client.get_device_fleet_report(
    DeviceFleetName="sample-fleet-name"
)
```

- **Check for a heartbeat.**

Each device within a fleet periodically generates a signal, or “heartbeat”. The heartbeat can be used to check that the device is communicating with Edge Manager. If the timestamp of the last heartbeat is not being updated, the device may be failing.

Check the last heartbeat made by a device with the `DescribeDevice` API. Specify the name of the device and the fleet to which the edge device belongs.

```
sagemaker_client.describe_device(
    DeviceName="sample-device-1",
    DeviceFleetName="sample-fleet-name"
)
```

## Package Model

SageMaker Edge Manager packaging jobs take Amazon SageMaker Neo-compiled models and make any changes necessary to deploy the model with the inference engine, Edge Manager agent.

### Topics

- [Prerequisites \(p. 2107\)](#)
- [Package a Model \(Amazon SageMaker Console\) \(p. 2109\)](#)
- [Package a Model \(Boto3\) \(p. 2112\)](#)

## Prerequisites

To package a model, you must do the following:

1. **Compile your machine learning model with SageMaker Neo.**

If you have not already done so, compile your model with SageMaker Neo. For more information on how to compile your model, see [Compile and Deploy Models with Neo](#). If you are first-time user of SageMaker Neo, go through [Getting Started with Neo Edge Devices](#).

**2. Get the name of your compilation job.**

Provide the name of the compilation job name you used when you compiled your model with SageMaker Neo. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/> and choose **Compilation jobs** to find a list of compilations that have been submitted to your Amazon account. The names of submitted compilation jobs are in the **Name** column.

**3. Get your IAM ARN.**

You need an Amazon Resource Name (ARN) of an IAM role that you can use to download and upload the model and contact SageMaker Neo.

Use one of the following methods to get your IAM ARN:

- **Programmatically with the SageMaker Python SDK**

```
import sagemaker

# Initialize SageMaker Session object so you can interact with Amazon resources
sess = sagemaker.Session()

# Get the role ARN
role = sagemaker.get_execution_role()

print(role)
>> arn:aws:iam::<your-aws-account-id>:role/<your-role-name>
```

For more information about using the SageMaker Python SDK, see the [SageMaker Python SDK API](#).

- **Using the Amazon Identity and Access Management (IAM) console**

Navigate to the IAM console at <https://console.amazonaws.cn/iam/>. In the **IAM Resources** section, choose **Roles** to view a list of roles in your Amazon account. Select or create a role that has **AmazonSageMakerFullAccess**, **AWSIoTFullAccess**, and **AmazonS3FullAccess**.

For more information on IAM, see [What is IAM?](#)

**4. Have an S3 bucket URI.**

You need to have at least one Amazon Simple Storage Service (Amazon S3) bucket URI to store your Neo-compiled model, the output of the Edge Manager packaging job, and sample data from your device fleet.

Use one of the following methods to create an Amazon S3 bucket:

- **Programmatically with the SageMaker Python SDK**

You can use the default Amazon S3 bucket during a session. A default bucket is created based on the following format: `sagemaker-{region}-{aws-account-id}`. To create a default bucket with the SageMaker Python SDK, use the following:

```
import sagemaker

session=sagemaker.create_session()

bucket=session.default_bucket()
```

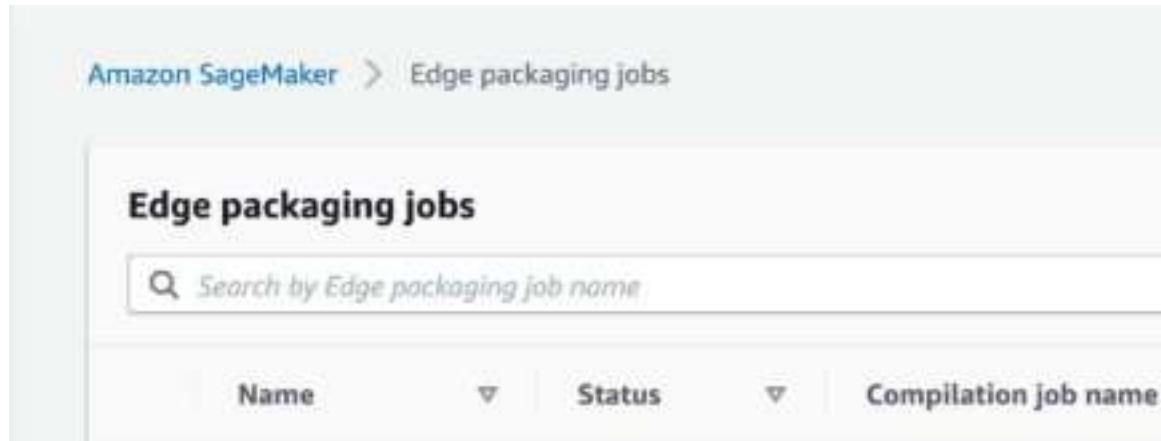
- **Using the Amazon S3 console**

Open the Amazon S3 console at <https://console.amazonaws.cn/s3/> and see [How do I create an S3 Bucket?](#) for step-by-step instructions.

## Package a Model (Amazon SageMaker Console)

You can create a SageMaker Edge Manager packaging job using the SageMaker console at <https://console.amazonaws.cn/sagemaker/>. Before continuing, make sure you have satisfied the [Prerequisites \(p. 2107\)](#).

1. In the SageMaker console, choose **Edge Inference** and then choose **Edge packaging jobs**, as shown in the following image.



2. On the **Job properties** page, enter a name for your packaging job under **Edge packaging job name**. Note that Edge Manager packaging job names are case-sensitive. Name your model and give it a version: enter this under **Model name** and **Model version**, respectively.
3. Next, select an **IAM role**. You can chose a role or let Amazon create a role for you. You can optionally specify a **resource key ARN** and **job tags**.
4. Choose **Next**.

## Job properties

Edge packaging job name

sample-edge-package

63 characters max

Model name

sample-model

128 characters max

Model version

1

128 characters max

IAM role

Amazon SageMaker Edge requires permissions to create an [AmazonSageMakerFullAccess](#) IAM policy attached.

Admin

---

Resource key ARN - *optional*

Enter the resource key to encrypt the EBS volume the

- Specify the name of the compilation job you used when compiling your model with SageMaker Neo in the **Compilation job name** field. Choose **Next**.

**Model source**

Specify the name of your SageMaker Neo compilation job in the field below. SageMaker Edge needs to know the model artifacts.

**Compilation job name**

Specify the name of the compilation job you used when compiling your model with SageMaker Neo. Compile your code before moving on if you have not done so yet. [Manage compilation jobs](#)

getting-started-demo

**Cancel**

- On the **Output configuration** page, enter the Amazon S3 bucket URI in which you want to store the output of the packaging job.

**Output configuration**

Use the fields below to specify the S3 bucket URI where you want devices to store sample data. You can also (optionally) specify a KMS key.

**S3 bucket URI**

Enter your S3 bucket URI where you want devices to store sample data.

s3://bucket-example/sagemaker-edge/device-output

To find a path, [go to Amazon S3](#)

**Encryption key - optional**

Encrypt your data. Choose an existing KMS key or enter a key's ARN.

No Custom Encryption

**Cancel**

The **Status** column on the **Edge packaging** jobs page should read **IN PROGRESS**. Once the packaging job is complete, the status updates to **COMPLETED**.

Selecting a packaging job directs you to that job's settings. The **Job settings** section displays the job name, ARN, status, creation time, last modified time, duration of the packaging job, and role ARN.

The **Input configuration** section displays the location of the model artifacts, the data input configuration, and the machine learning framework of the model.

The **Output configuration** section displays the output location of the packaging job, the target device for which the model was compiled, and any tags you created.

7. Choose the name of your device fleet to be redirected to the device fleet details. This page displays the name of the device fleet, ARN, description (if you provided one), date the fleet was created, last time the fleet was modified, Amazon S3 bucket URI, Amazon KMS key ID (if provided), Amazon IoT alias (if provided), and IAM role. If you added tags, they appear in the **Device fleet tags** section.

## Package a Model (Boto3)

You can create a SageMaker Edge Manager packaging job with the Amazon SDK for Python (Boto3). Before continuing, make sure you have satisfied the [Prerequisites \(p. 2107\)](#).

To request an edge packaging job, use `CreateEdgePackagingJob`. You need to provide a name to your edge packaging job, the name of your SageMaker Neo compilation job, your role Amazon resource name (ARN), a name for your model, a version for your model, and the Amazon S3 bucket URI where you want to store the output of your packaging job. Note that Edge Manager packaging job names and SageMaker Neo compilation job names are case-sensitive.

```
# Import Amazon SDK for Python (Boto3)
import boto3

# Create Edge client so you can submit a packaging job
sagemaker_client = boto3.client("sagemaker", region_name='aws-region')

sagemaker_client.create_edge_packaging_job(
    EdgePackagingJobName="edge-packaging-name",
    CompilationJobName="neo-compilation-name",
    RoleArn="arn:aws:iam::999999999999:role/rolename",
    ModelName="sample-model-name",
    ModelVersion="model-version",
    OutputConfig={
        "S3OutputLocation": "s3://your-bucket/",
    }
)
```

You can check the status of an edge packaging job using `DescribeEdgePackagingJob` and providing the case-sensitive edge packaging job name:

```
response = sagemaker_client.describe_edge_packaging_job(
    EdgePackagingJobName="edge-packaging-name")
```

This returns a dictionary that can be used to poll the status of the packaging job:

```
# Optional - Poll every 30 sec to check completion status
import time

while True:
    response = sagemaker_client.describe_edge_packaging_job(
        EdgePackagingJobName="edge-packaging-name")

    if response['EdgePackagingJobStatus'] == 'Completed':
        break
    elif response['EdgePackagingJobStatus'] == 'Failed':
        raise RuntimeError('Packaging job failed')
```

```
print('Packaging model... ')
time.sleep(30)
print('Done!')
```

For a list of packaging jobs, use `ListEdgePackagingJobs`. You can use this API to search for a specific packaging job. Provide a partial name to filter packaging job names for `NameContains`, a partial name for `ModelNameContains` to filter for jobs in which the model name contains the name you provide. Also specify with which column to sort with `SortBy`, and by which direction to sort for `SortOrder` (either `Ascending` or `Descending`).

```
sagemaker_client.list_edge_packaging_jobs(
    "NameContains": "sample",
    "ModelNameContains": "sample",
    "SortBy": "column-name",
    "SortOrder": "Descending"
)
```

To stop a packaging job, use `StopEdgePackagingJob` and provide the name of your edge packaging job.

```
sagemaker_client.stop_edge_packaging_job(
    EdgePackagingJobName="edge-packaging-name"
)
```

For a full list of Edge Manager APIs, see the [Boto3 documentation](#).

## Edge Manager Agent

The Edge Manager agent is an inference engine for your edge devices. Use the agent to make predictions with models loaded onto your edge devices. The agent also collects model metrics and captures data at specific intervals. Sample data is stored in your Amazon S3 bucket.

There are two methods of installing and deploying the Edge Manager agent onto your edge devices:

1. Download the agent as a binary from the Amazon S3 release bucket. For more information, see [Download and Set Up Edge Manager Agent Manually \(p. 2113\)](#).
2. Use the Amazon IoT Greengrass V2 console or the Amazon CLI to deploy `aws.greengrass.SageMakerEdgeManager`. See [Create Amazon IoT Greengrass V2 Components \(p. 2119\)](#).

## Download and Set Up Edge Manager Agent Manually

Download the Edge Manager agent based on your operating system, architecture, and your Amazon Region. The agent is periodically updated, so you have the option to choose your agent based on release dates and versions. Once you have the agent, create a JSON configuration file. Specify the device IoT thing name, fleet name, device credentials, and other key-value pairs. See [Running SageMaker Edge Manager Agent \(p. 2115\)](#) for full a list of keys you must specify in the configuration file. You can run the agent as an executable binary or link against it as a Dynamic Shared Object .

### How the Agent Works

The agent runs on the CPU of your devices. The agent runs inference on the framework and hardware of the target device you specified during the compilation job. For example, if you compiled your model for the Jetson Nano, the agent supports the GPU in the provided [Deep Learning Runtime \(DLR\)](#).

The agent is released in binary format for supported operating systems. Check that your operating system is supported and meets the minimum OS requirement in the following table:

Linux

**Version:** Ubuntu 18.04

**Supported Binary Formats:** x86-64 bit (ELF binary) and ARMv8 64 bit (ELF binary)

Windows

**Version:** Windows 10 version 1909

**Supported Binary Formats:** x86-32 bit (DLL) and x86-64 bit (DLL)

## Installing Edge Manager Agent

To use Edge Manager agent, you first must obtain the release artifacts and a Root Certificate. The release artifacts are stored in an Amazon S3 bucket in the us-west-2 Region. To download the artifacts, specify your operating system (<OS>) and the VERSION.

Based on your operating system, replace <os> with one of the following:

Windows 32-bit	Windows 64-bit	Linux x86-64	Linux ARMv8
windows-x86	windows-x64	linux-x64	linux-armv8

The VERSION is broken into three components: <MAJOR\_VERSION>. <YYYY-MM-DD>-<SHA-7>, where:

- **MAJOR\_VERSION:** The release version. The release version is currently set to 1.
- <YYYY-MM-DD>: The time stamp of the artifacts release.
- **SHA-7:** The repository commit ID from which the release is built.

You must provide the MAJOR\_VERSION and the time stamp in <YYYY-MM-DD> format. We suggest you use the latest artifact release time stamp. Use the following to get the latest time stamp.

Run the following in your command line to get the latest time stamp. Replace <os> with your operating system:

```
aws s3 ls s3://sagemaker-edge-release-store-us-west-2-<OS>/Releases/ | sort -r
```

For example, if you have a Windows 32-bit OS, run:

```
aws s3 ls s3://sagemaker-edge-release-store-us-west-2-windows-x86/Releases/ | sort -r
```

This returns:

```
2020-12-01 23:33:36 0
PRE 1.20201218.81f481f/
PRE 1.20201207.02d0e97/
```

The return output in this example shows two release artifacts. The first release artifact file notes that the release version has a major release version of 1, a time stamp of 20201218 (in <YYYY-MM-DD> format), and a 81f481f SHA-7 commit ID.

**Note**

The preceding command assumes you have configured the Amazon Command Line Interface. For more information, about how to configure the settings that the Amazon CLI uses to interact with Amazon, see [Configuring the Amazon CLI](#).

Based on your operating system, use the following commands to install the artifacts:

Windows 32-bit

```
aws s3 cp s3://sagemaker-edge-release-store-us-west-2-windows-x86/
Releases/<VERSION>/<VERSION>.zip .
aws s3 cp s3://sagemaker-edge-release-store-us-west-2-windows-x86/Releases/<VERSION>/
sha256_hex.shasum .
```

Windows 64-bit

```
aws s3 cp s3://sagemaker-edge-release-store-us-west-2-windows-x64/
Releases/<VERSION>/<VERSION>.zip .
aws s3 cp s3://sagemaker-edge-release-store-us-west-2-windows-x64/Releases/<VERSION>/
sha256_hex.shasum .
```

Linux x86-64

```
aws s3 cp s3://sagemaker-edge-release-store-us-west-2-linux-x64/
Releases/<VERSION>/<VERSION>.tgz .
aws s3 cp s3://sagemaker-edge-release-store-us-west-2-linux-x64/Releases/<VERSION>/
sha256_hex.shasum .
```

Linux ARMv8

```
aws s3 cp s3://sagemaker-edge-release-store-us-west-2-linux-armv8/
Releases/<VERSION>/<VERSION>.tgz .
aws s3 cp s3://sagemaker-edge-release-store-us-west-2-linux-armv8/Releases/<VERSION>/
sha256_hex.shasum .
```

You also must download a Root Certificate. This certificate validates model artifacts signed by Amazon before loading them onto your edge devices.

Replace <OS> corresponding to your platform from the list of supported operation systems and replace <REGION> with your Amazon Region.

```
aws s3 cp s3://sagemaker-edge-release-store-us-west-2-<OS>/
Certificates/<REGION>/<REGION>.pem .
```

## Running SageMaker Edge Manager Agent

You can run SageMaker Edge Manager agent can as a standalone process in the form of an Executable and Linkable Format (ELF) executable binary or you can link against it as a Dynamic Shared Object (.dll). Running as a standalone executable binary is the preferred mode and is supported on Linux. Running as a shared object (.dll) is supported on Windows.

On Linux, we recommend that you run the binary via a service that's a part of your initialization (`init`) system. If you want to run the binary directly, you can do so in a terminal as shown in the following example. If you have a modern OS, there are no other installations necessary prior to running the agent, since all the requirements are statically built into the executable. This gives you flexibility to run the agent on the terminal, as a service, or within a container.

To run the agent, first create a JSON configuration file. Specify the following key-value pairs:

- **deviceName** : The name of the device. This device name needs to be registered along with the device fleet in the SageMaker Edge Manager console.
- **deviceFleetName**: The name of the fleet to which the device belongs.
- **region**: The Amazon Region.
- **rootCertsPath**: The absolute folder path to root certificates.
- **provider**: Use "Aws".
- **awsCACertFile**: The absolute path to Amazon Root CA certificate (AmazonRootCA1.pem).
- **awsCertFile**: The absolute path to Amazon IoT signing root certificate (\*.pem.crt).
- **awsCertPKFile**: The absolute path to Amazon IoT private key. (\*.pem.key).
- **awsIoTCredEndpoint**: The Amazon IoT credentials endpoint (*identifier*.iot.*region*.amazonaws.com). See [Connecting devices to Amazon IoT](#) for more information.
- **captureDataDestination**: The destination for uploading capture data. Select either Cloud or Disk.
- **S3BucketName**: The name of your Amazon S3 bucket (not the Amazon S3 bucket URI). The bucket must have a `sagemaker` string within its name.
- **folderPrefix**: The Amazon S3 folder name where you want captured data to upload.
- **captureDataBufferSize**: The capture data buffer size. Integer value.
- **captureDataBatchSize**: The capture data batch size. Integer value.
- **pushPeriodSeconds**: The capture data push period in seconds.
- **base64EmbedLimit**: The limit for uploading capture data in bytes. Integer value.
- **logVerbose** (Optional): Set debug log. Boolean. Select either True or False.

Your configuration file should look similar to the following (with your specific values specified):

```
{
    "sagemaker_edge_core_device_name": "device-name",
    "sagemaker_edge_core_device_fleet_name": "fleet-name",
    "sagemaker_edge_core_region": "region",
    "sagemaker_edge_core_root_certs_path": "<Absolute path to root certificates>",
    "sagemaker_edge_provider_provider": "Aws",
    "sagemaker_edge_provider_aws_ca_cert_file": "<Absolute path to Amazon Root CA certificate>/AmazonRootCA1.pem",
    "sagemaker_edge_provider_aws_cert_file": "<Absolute path to Amazon IoT signing root certificate>/device.pem.crt",
    "sagemaker_edge_provider_aws_cert_pk_file": "<Absolute path to Amazon IoT private key.>/private.pem.key",
    "sagemaker_edge_provider_aws_iot_cred_endpoint": "https://<Amazon IoT Endpoint Address>",
    "sagemaker_edge_core_capture_data_destination": "Cloud",
    "sagemaker_edge_provider_s3_bucket_name": "sagemaker-bucket-name",
    "sagemaker_edge_core_folder_prefix": "Amazon S3 folder prefix",
    "sagemaker_edge_core_capture_data_buffer_size": 30,
    "sagemaker_edge_core_capture_data_batch_size": 10,
    "sagemaker_edge_core_capture_data_push_period_seconds": 4,
    "sagemaker_edge_core_capture_data_base64_embed_limit": 20,
    "sagemaker_edge_log_verbose": false
}
```

Included in the release artifact is a binary executable called `sagemaker_edge_agent_binary` in the `/bin` directory. To run the binary, use the `-a` flag to create a socket file descriptor (.sock) in a directory of your choosing and specify the path of the agent JSON config file you created with the `-c` flag.

```
./sagemaker_edge_agent_binary -a <ADDRESS_TO_SOCKET> -c <PATH_TO_CONFIG_FILE>
```

For example:

```
./sagemaker_edge_agent_binary -a /tmp/sagemaker_edge_agent_example.sock -c sagemaker_edge_config.json
```

In this example, a socket file descriptor named `sagemaker_edge_agent_example.sock` is created in the `/tmp` directory and points to a configuration file that is in the same working directory as the agent called `sagemaker_edge_config.json`.

## Deploy Model Package and Edge Manager Agent with Amazon IoT Greengrass

SageMaker Edge Manager integrates Amazon IoT Greengrass version 2 to simplify accessing, maintaining, and deploying the Edge Manager agent and model to your devices. Without Amazon IoT Greengrass V2, setting up your devices and fleets to use SageMaker Edge Manager requires you to manually copy the Edge Manager agent from an Amazon S3 release bucket. You use the agent to make predictions with models loaded onto your edge devices. With Amazon IoT Greengrass V2 and SageMaker Edge Manager integration, you can now use Amazon IoT Greengrass V2 components. Components are pre-built software modules that can connect your edge devices to Amazon services or third-party service via Amazon IoT Greengrass.

You must install the Amazon IoT Greengrass Core software onto your device(s) if you want to use Amazon IoT Greengrass V2 to deploy the SageMaker Edge Manager agent and your model. For more information about device requirements and how to set up your devices, see [Setting up Amazon IoT Greengrass core devices](#) in the Amazon IoT Greengrass documentation.

You use a pre-built component, an autogenerated component, and a custom component for your Edge Manager application. The public SageMaker Edge Manager component is maintained by SageMaker. The private component is autogenerated when you package your machine learning model with the `createEdgePackagingJob` API and specify "GreengrassV2Component" for the Edge Manager API field `PresetDeploymentType`. The third component is the inference application that is responsible for preprocessing and making inferences on your device. You must create this component. See either [???](#) (p. 2120) in the SageMaker Edge Manager documentation or [Create custom Amazon IoT Greengrass components](#) in the Amazon IoT Greengrass documentation for more information on how to create custom components.

### Prerequisites

SageMaker Edge Manager uses Amazon IoT Greengrass V2 to simplify the deployment of the Edge Manager agent, your machine learning models, and your inference application to your device(s) with the use of components. To make it easier to maintain your Amazon IAM roles, Edge Manager allows you to reuse your existing Amazon IoT role alias. If you do not have one yet, Edge Manager generates a role alias as part of the Edge Manager packaging job. You no longer need to associate a role alias generated from the SageMaker Edge Manager packaging job with your Amazon IoT Role.

Before you start, you must:

1. Install the Amazon IoT Greengrass Core software. For detailed information, see [Install the Amazon IoT Greengrass Core software](#).
2. Set up Amazon IoT Greengrass V2. For more information, see [Install Amazon IoT Greengrass Core software with manual resource provisioning](#).

#### Note

- Make sure the Amazon IoT thing name is all lowercase.
- IAM Role starts with `SageMaker*`

3. Attach the following permission and inline policy to the IAM role created during Amazon IoT Greengrass V2 setup.

- Navigate to the IAM console <https://console.amazonaws.cn/iam/>.
- Search for the role you created by typing in the role name in the **Search** field.
- Choose your role.
- Next, choose **Attach policies**.
- Search for **AmazonSageMakerEdgeDeviceFleetPolicy**.
- Select **AmazonSageMakerFullAccess** (This is an optional step that makes it easier for you to reuse this IAM role in model compilation and packaging).
- Select **Add inline policy**.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "GreengrassComponentAccess",  
            "Effect": "Allow",  
            "Action": [  
                "greengrass>CreateComponentVersion",  
                "greengrassDescribeComponent"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

- Click **Attach policy**.
- Select **Trust relationship**.
- Click **Edit trust relationship**.
- Replace the content with the following.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "credentials.iot.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        },  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Service": "sagemaker.amazonaws.com"  
            },  
            "Action": "sts:AssumeRole"  
        }  
    ]  
}
```

4. Create an Edge Manager device fleet. For information on how to create a fleet, see [Set Up Devices and Fleets \(p. 2100\)](#).
5. Register your device with the same name as your Amazon IoT thing name created during the Amazon IoT Greengrass V2 setup.
6. Create at least one custom private Amazon IoT Greengrass component. This component is the application that runs inference on the device. See [Create a Hello World custom component \(p. 2120\)](#)

### Note

- The SageMaker Edge Manager and Amazon IoT Greengrass integration only works for Amazon IoT Greengrass v2.
- Both your Amazon IoT thing name and Edge Manager device name must be the same.
- SageMaker Edge Manager does not load local Amazon IoT certificates and call the Amazon IoT credential provider endpoint directly. Instead, SageMaker Edge Manager uses the Amazon IoT Greengrass v2 TokenExchangeService and it fetches a temporary credential from a TES endpoint.

## Create Amazon IoT Greengrass V2 Components

Amazon IoT Greengrass uses *components*, a software module that is deployed to and runs on a Amazon IoT Greengrass core device. You will need (at a minimum) three components:

1. A public Edge Manager Agent Amazon IoT Greengrass component.
2. A model component that is autogenerated when you package your machine learning model with either the Amazon SDK for Python (Boto3) API or with the SageMaker console.
3. A private, custom component for the inference application.

The Edge Manager Agent Amazon IoT Greengrass component (1) deploys the Edge Manager Agent binary.

The model component (2) is autogenerated when you create an edge packaging job with either the Amazon SDK for Python (Boto3) API or the SageMaker console. For information on how to generate the model component, see [Autogenerated Component \(p. 2119\)](#).

The private custom component (3) is the application that you use to implement the Edge Manager Agent client application, as well as do any preprocessing and post-processing of the inference results. For more information about how to create a custom component, see [Autogenerated Component \(p. 2119\)](#) or [Create custom Amazon IoT Greengrass components](#).

### Autogenerated Component

Generate the model component with the [CreateEdgePackagingJob](#) API and specify `GreengrassV2Component` for the SageMaker Edge Manager packaging job API field `PresetDeploymentType`. When you call the `CreateEdgePackagingJob` API, Edge Manager takes your SageMaker Neo-compiled model in Amazon S3 and creates a model component. The model component is automatically stored in your account. You can view any of your components by navigating to the Amazon IoT console <https://console.amazonaws.cn/iot/>. Select **Greengrass** and then select **Core** devices. The page has a list of Amazon IoT Greengrass core devices associated with your account. If a model component name is not specified in `PresetDeploymentConfig`, the default name generated consists of "SagemakerEdgeManager" and the name of your SageMaker Edge Manager packaging job. The following example demonstrates how to specify Edge Manager to create a Amazon IoT Greengrass V2 component with the `CreateEdgePackagingJob` API.

```
import sagemaker
import boto3

# Create a SageMaker client object to make it easier to interact with other AWS services.
sagemaker_client = boto3('sagemaker', region=<YOUR_REGION>

# Replace with your IAM Role ARN
sagemaker_role_arn="arn:aws:iam::<account>:role/*"

# Replace string with the name of your already created S3 bucket.
bucket='edge-manager-demo-bucket'
```

```
# Specify a name for your edge packaging job.  
edge_packaging_name="edge_packag_job_demo"  
  
# Replace the following string with the name you used for the SageMaker Neo compilation  
# job.  
compilation_job_name="getting-started-demo"  
  
# The name of the model and the model version.  
model_name="sample-model"  
model_version="1.1"  
  
# Output directory in S3 where you want to store the packaged model.  
packaging_output_dir = 'packaged_models'  
packaging_s3_output = 's3://{}{}'.format(bucket, packaging_output_dir)  
  
# The name you want your Greengrass component to have.  
component_name="SagemakerEdgeManager"+edge_packaging_name  
  
sagemaker_client.create_edge_packaging_job(  
    EdgePackagingJobName=edge_packaging_name,  
    CompilationJobName=compilation_job_name,  
    RoleArn=sagemaker_role_arn,  
    ModelName=model_name,  
    ModelVersion=model_version,  
    OutputConfig={  
        "S3OutputLocation": packaging_s3_output,  
        "PresetDeploymentType": "GreengrassV2Component",  
        "PresetDeploymentConfig": {"ComponentName": component_name}  
    }  
)
```

You can also create the autogenerated component with the SageMaker console. Follow steps 1-6 in [Package a Model \(Amazon SageMaker Console\) \(p. 2109\)](#)

Enter the Amazon S3 bucket URI where you want to store the output of the packaging job and optional encryption key.

Complete the following to create the model component:

1. Choose **Preset deployment**.
2. Specify the name of the component for the **Component name** field.
3. Optionally, provide a description of the component, a component version, the platform OS, or the platform architecture for the **Component description**, **Component version**, **Platform OS**, and **Platform architecture**, respectively.
4. Choose **Submit**.

### Create a Hello World custom component

The custom application component is used to perform inference on the edge device. The component is responsible for loading models to SageMaker Edge Manager, invoking the Edge Manager agent for inference, and unloading the model when the component is shut down. Before you create your component, ensure the agent and application can communicate with SageMaker Edge Manager. To do this, configure gRPC. The SageMaker Edge Manager agent uses methods defined in Protobuf Buffers and the gRPC server to establish communication with the client application on the edge device and the cloud.

To use gRPC, you must:

1. Create a gRPC stub using the .proto file provided when you download the Edge Manager agent from Amazon S3 release bucket.
2. Write client code with the language you prefer.

You do not need to define the service in a .proto file (1). The service .proto files are included in the compressed TAR file when you download the SageMaker Edge Manager agent release binary from the Amazon S3 release bucket.

Install gRPC and other necessary tools on your host machine and create gRPC stubs `agent_pb2_grpc.py` and `agent_pb2.py` in Python. Make sure you have `agent.proto` in your local directory.

```
%%bash
pip install grpcio
pip install grpcio-tools
python3 -m grpc_tools.protoc --proto_path=. --python_out=. --grpc_python_out=. agent.proto
```

The preceding code generates the gRPC client and server interfaces from your .proto service definition (2). In other words, it creates the gRPC model in Python. The API folder contains the Protobuf specification for communicating with the agent.

Next, use the gRPC API to write a client and server for your service (2). The following example script, `edge_manager_python_example.py`, uses Python to load, list, and unload a yolov3 model to the edge device.

```
import grpc
from PIL import Image
import agent_pb2
import agent_pb2_grpc
import os

model_path = '<PATH-TO-SagemakerEdgeManager-COMPONENT>'

agent_socket = 'unix:///tmp/aws.greengrass.SageMakerEdgeManager.sock'
agent_channel = grpc.insecure_channel(agent_socket, options=(('grpc.enable_http_proxy', 0),))
agent_client = agent_pb2_grpc.AgentStub(agent_channel)

def list_models():
    return agent_client.ListModels(agent_pb2.ListModelsRequest())

def list_model_tensors(models):
    return [
        model.name: {
            'inputs': model.input_tensor_metadata,
            'outputs': model.output_tensor_metadata
        }
        for model in list_models().models
    ]

def load_model(model_name, model_path):
    load_request = agent_pb2.LoadModelRequest()
    load_request.url = model_path
    load_request.name = model_name
    return agent_client.LoadModel(load_request)

def unload_model(name):
    unload_request = agent_pb2.UnLoadModelRequest()
    unload_request.name = name
    return agent_client.UnLoadModel(unload_request)
```

```

def predict_image(model_name, image_path):
    image_tensor = agent_pb2.Tensor()
    image_tensor.byte_data = Image.open(image_path).tobytes()
    image_tensor_metadata = list_model_tensors(list_models())[model_name]['inputs'][0]
    image_tensor.tensor_metadata.name = image_tensor_metadata.name
    image_tensor.tensor_metadata.data_type = image_tensor_metadata.data_type
    for shape in image_tensor_metadata.shape:
        image_tensor.tensor_metadata.shape.append(shape)
    predict_request = agent_pb2.PredictRequest()
    predict_request.name = model_name
    predict_request.tensors.append(image_tensor)
    predict_response = agent_client.Predict(predict_request)
    return predict_response

def main():
    try:
        unload_model('your-model')
    except:
        pass

    print('LoadModel...', end='')
    try:
        load_model('your-model', model_path)
        print('done.')
    except Exception as e:
        print()
        print(e)
        print('Model already loaded!')

    print('ListModels...', end='')
    try:
        print(list_models())
        print('done.')
    except Exception as e:
        print()
        print(e)
        print('List model failed!')

    print('Unload model...', end='')
    try:
        unload_model('your-model')
        print('done.')
    except Exception as e:
        print()
        print(e)
        print('unload model failed!')

if __name__ == '__main__':
    main()

```

Ensure `model_path` points to the name of the Amazon IoT Greengrass component containing the model if you use the same client code example.

You can create your Amazon IoT Greengrass V2 Hello World component once you have generated your gRPC stubs and you have your Hello World code ready. To do so:

- Upload your `edge_manager_python_example.py`, `agent_pb2_grpc.py`, and `agent_pb2.py` to your Amazon S3 bucket and note down their Amazon S3 path.
- Create a private component in the Amazon IoT Greengrass V2 console and define the recipe for your component. Specify the Amazon S3 URI to your Hello World application and gRPC stub in the following recipe.

```
---  
RecipeFormatVersion: 2020-01-25  
ComponentName: com.sagemaker.edgePythonExample  
ComponentVersion: 1.0.0  
ComponentDescription: Sagemaker Edge Manager Python example  
ComponentPublisher: Amazon Web Services, Inc.  
ComponentDependencies:  
    aws.greengrass.SageMakerEdgeManager:  
        VersionRequirement: '>=1.0.0'  
        DependencyType: HARD  
Manifests:  
    - Platform:  
        os: linux  
        architecture: "/amd64|x86/"  
Lifecycle:  
    install: |-  
        apt-get install python3-pip  
        pip3 install grpcio  
        pip3 install grpcio-tools  
        pip3 install protobuf  
        pip3 install Pillow  
    run:  
        script: |-  
            python3 {{artifacts:path}}/edge_manager_python_example.py  
Artifacts:  
    - URI: <code-s3-path>  
    - URI: <pb2-s3-path>  
    - URI: <pb2-grpc-s3-path>
```

For detailed information about creating a Hello World recipe, see [Create your first component](#) in the Amazon IoT Greengrass documentation.

## Deploy Components to Your Device

Deploy your components with the Amazon IoT console or with the Amazon CLI.

### To deploy your components (console)

Deploy your Amazon IoT Greengrass components with the Amazon IoT console.

1. In the Amazon IoT Greengrass console at <https://console.amazonaws.cn/iot/> navigation menu, choose **Deployments**.
2. On the **Components** page, on the **Public components** tab, choose `aws.greengrass.SageMakerEdgeManager`.
3. On the `aws.greengrass.SageMakerEdgeManager` page, choose **Deploy**.
4. From **Add to deployment**, choose one of the following:
  - a. To merge this component to an existing deployment on your target device, choose **Add to existing deployment**, and then select the deployment that you want to revise.
  - b. To create a new deployment on your target device, choose **Create new deployment**. If you have an existing deployment on your device, choosing this step replaces the existing deployment.
5. On the **Specify target** page, do the following:
  - a. Under **Deployment information**, enter or modify the friendly name for your deployment.
  - b. Under **Deployment targets**, select a target for your deployment, and choose **Next**. You cannot change the deployment target if you are revising an existing deployment.
6. On the **Select components** page, under **My components**, choose:
  - com.<CUSTOM-COMPONENT-NAME>

- `aws.greengrass.SageMakerEdgeManager`
  - `SagemakerEdgeManager.<YOUR-PACKAGING-JOB>`
7. On the **Configure components** page, choose `com.greengrass.SageMakerEdgeManager`, and do the following.
    - a. Choose **Configure component**.
    - b. Under **Configuration update**, in **Configuration to merge**, enter the following configuration.

```
{
  "DeviceFleetName": "device-fleet-name",
  "BucketName": "DOC-EXAMPLE-BUCKET"
}
```

Replace `device-fleet-name` with the name of the edge device fleet that you created, and replace `DOC-EXAMPLE-BUCKET` with the name of the Amazon S3 bucket that is associated with your device fleet.

- c. Choose **Confirm**, and then choose **Next**.
8. On the **Configure advanced settings** page, keep the default configuration settings, and choose **Next**.
9. On the **Review** page, choose **Deploy**.

### To deploy your components (Amazon CLI)

1. Create a `deployment.json` file to define the deployment configuration for your SageMaker Edge Manager components. This file should look like the following example.

```
{
  "targetArn": "targetArn",
  "components": {
    "aws.greengrass.SageMakerEdgeManager": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
        "merge": {
          "DeviceFleetName": "device-fleet-name",
          "BucketName": "DOC-EXAMPLE-BUCKET"
        }
      }
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {}
    },
    "com.greengrass.SageMakerEdgeManager.ImageClassification.Model": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {}
    }
  }
}
```

- In the `targetArn` field, replace `targetArn` with the Amazon Resource Name (ARN) of the thing or thing group to target for the deployment, in the following format:
  - Thing: `arn:aws:iot:<region>:<account-id>:thing/<thingName>`
  - Thing group: `arn:aws:iot:<region>:<account-id>:thinggroup/<thingGroupName>`
- In the `merge` field, replace `device-fleet-name` with the name of the edge device fleet that you created, and replace `DOC-EXAMPLE-BUCKET` with the name of the Amazon S3 bucket that is associated with your device fleet.

- Replace the component versions for each component with the latest available version.
2. Run the following command to deploy the components on the device:

```
$ aws greengrassv2 create-deployment \
--cli-input-json file://path/to/deployment.json
```

The deployment can take several minutes to complete. In the next step, check the component log to verify that the deployment completed successfully and to view the inference results.

For more information about deploying components to individual devices or groups of devices, see [Deploy Amazon IoT Greengrass components to devices](#).

## Manage Model

The Edge Manager agent can load multiple models at a time and make inference with loaded models on edge devices. The number of models the agent can load is determined by the available memory on the device. The agent validates the model signature and loads into memory all the artifacts produced by the edge packaging job. This step requires all the required certificates described in previous steps to be installed along with rest of the binary installation. If the model's signature cannot be validated, then loading of the model fails with appropriate return code and reason.

SageMaker Edge Manager agent provides a list of Model Management APIs that implement control plane and data plane APIs on edge devices. Along with this documentation, we recommend going through the sample client implementation which shows canonical usage of the below described APIs.

The `proto` file is available as a part of the release artifacts (inside the release tarball). In this doc, we list and describe the usage of APIs listed in this `proto` file.

### Note

There is one-to-one mapping for these APIs on Windows release and a sample code for an application implement in C# is shared with the release artifacts for Windows. Below instructions are for running the Agent as a standalone process, applicable for to the release artifacts for Linux.

Extract the archive based on your OS. Where `VERSION` is broken into three components: `<MAJOR_VERSION>. <YYYY-MM-DD>-<SHA-7>`. See [Installing Edge Manager Agent \(p. 2114\)](#) for information on how to obtain the release version (`<MAJOR_VERSION>`), time stamp of the release artifact (`<YYYY-MM-DD>`), and the repository commit ID (`SHA-7`)

### Linux

The zip archive can be extracted with the command:

```
tar -xvzf <VERSION>.tgz
```

### Windows

The zip archive can be extracted with the UI or command:

```
unzip <VERSION>.tgz
```

The release artifact hierarchy (after extracting the `tar/zip` archive) is shown below. The agent `proto` file is available under `api/`.

```
0.20201205.7ee4b0b
### bin
```

```

#           ### sagemaker_edge_agent_binary
#           ### sagemaker_edge_agent_client_example
### docs
### api
#           ### agent.proto
### attributions
#           ### agent.txt
#           ### core.txt
### examples
### ipc_example
### CMakeLists.txt
### sagemaker_edge_client.cc
### sagemaker_edge_client_example.cc
### sagemaker_edge_client.hh
### sagemaker_edge.proto
### README.md
### shm.cc
### shm.hh
### street_small.bmp

```

### Topics

- [Load Model \(p. 2126\)](#)
- [Unload Model \(p. 2127\)](#)
- [List Models \(p. 2128\)](#)
- [Describe Model \(p. 2128\)](#)
- [Capture Data \(p. 2129\)](#)
- [Get Capture Status \(p. 2130\)](#)
- [Predict \(p. 2131\)](#)

## Load Model

The Edge Manager agent supports loading multiple models. This API validates the model signature and loads into memory all the artifacts produced by the `EdgePackagingJob` operation. This step requires all the required certificates to be installed along with rest of the agent binary installation. If the model's signature cannot be validated then this step fails with appropriate return code and error messages in the log.

```

// perform load for a model
// Note:
// 1. currently only local filesystem paths are supported for loading models.
// 2. multiple models can be loaded at the same time, as limited by available device memory
// 3. users are required to unload any loaded model to load another model.
// Status Codes:
// 1. OK - load is successful
// 2. UNKNOWN - unknown error has occurred
// 3. INTERNAL - an internal error has occurred
// 4. NOT_FOUND - model doesn't exist at the url
// 5. ALREADY_EXISTS - model with the same name is already loaded
// 6. RESOURCE_EXHAUSTED - memory is not available to load the model
// 7. FAILED_PRECONDITION - model is not compiled for the machine.
//
rpc LoadModel(LoadModelRequest) returns (LoadModelResponse);

```

### Input

```

// 
// request for LoadModel rpc call
// 

```

```
message LoadModelRequest {
    string url = 1;
    string name = 2; // Model name needs to match regex "^[a-zA-Z0-9](-*[a-zA-Z0-9])*$"
}
```

#### Output

```
//
// response for LoadModel rpc call
//
message LoadModelResponse {
    Model model = 1;
}

//
// Model represents the metadata of a model
// url - url representing the path of the model
// name - name of model
// input_tensor_metadata - TensorMetadata array for the input tensors
// output_tensor_metadata - TensorMetadata array for the output tensors
//
// Note:
// 1. input and output tensor metadata could empty for dynamic models.
//
message Model {
    string url = 1;
    string name = 2;
    repeated TensorMetadata input_tensor_metadata = 3;
    repeated TensorMetadata output_tensor_metadata = 4;
}
```

## Unload Model

Unloads a previously loaded model. It is identified via the model alias which was provided during `loadModel`. If the alias is not found or model is not loaded then returns error.

```
//
// perform unload for a model
// Status Codes:
// 1. OK - unload is successful
// 2. UNKNOWN - unknown error has occurred
// 3. INTERNAL - an internal error has occurred
// 4. NOT_FOUND - model doesn't exist
//
rpc UnLoadModel(UnLoadModelRequest) returns (UnLoadModelResponse);
```

#### Input

```
//
// request for UnLoadModel rpc call
//
message UnLoadModelRequest {
    string name = 1; // Model name needs to match regex "^[a-zA-Z0-9](-*[a-zA-Z0-9])*$"
}
```

#### Output

```
//
```

```
// response for UnLoadModel rpc call
//
message UnLoadModelResponse {}
```

## List Models

Lists all the loaded models and their aliases.

```
//
// lists the loaded models
// Status Codes:
// 1. OK - unload is successful
// 2. UNKNOWN - unknown error has occurred
// 3. INTERNAL - an internal error has occurred
//
rpc ListModels(ListModelsRequest) returns (ListModelsResponse);
```

### Input

```
//
// request for ListModels rpc call
//
message ListModelsRequest {}
```

### Output

```
//
// response for ListModels rpc call
//
message ListModelsResponse {
    repeated Model models = 1;
}
```

## Describe Model

Describes a model that is loaded on the agent.

```
//
// Status Codes:
// 1. OK - load is successful
// 2. UNKNOWN - unknown error has occurred
// 3. INTERNAL - an internal error has occurred
// 4. NOT_FOUND - model doesn't exist at the url
//
rpc DescribeModel(DescribeModelRequest) returns (DescribeModelResponse);
```

### Input

```
//
// request for DescribeModel rpc call
//
message DescribeModelRequest {
    string name = 1;
}
```

### Output

```

//  

// response for DescribeModel rpc call  

//  

message DescribeModelResponse {  

    Model model = 1;  

}

```

## Capture Data

Allows the client application to capture input and output tensors in Amazon S3 bucket, and optionally the auxiliary. The client application is expected to pass a unique capture ID along with each call to this API. This can be later used to query status of the capture.

```

//  

// allows users to capture input and output tensors along with auxiliary data.  

// Status Codes:  

// 1. OK - data capture successfully initiated  

// 2. UNKNOWN - unknown error has occurred  

// 3. INTERNAL - an internal error has occurred  

// 5. ALREADY_EXISTS - capture initiated for the given capture_id  

// 6. RESOURCE_EXHAUSTED - buffer is full cannot accept any more requests.  

// 7. OUT_OF_RANGE - timestamp is in the future.  

// 8. INVALID_ARGUMENT - capture_id is not of expected format.  

//  

rpc CaptureData(CaptureDataRequest) returns (CaptureDataResponse);

```

### Input

```

enum Encoding {  

    CSV = 0;  

    JSON = 1;  

    NONE = 2;  

    BASE64 = 3;  

}  

//  

// AuxillaryData represents a payload of extra data to be capture along with inputs and  

// outputs of inference  

// encoding - supports the encoding of the data  

// data - represents the data of shared memory, this could be passed in two ways:  

// a. send across the raw bytes of the multi-dimensional tensor array  

// b. send a SharedMemoryHandle which contains the posix shared memory segment id and  

// offset in bytes to location of multi-dimensional tensor array.  

//  

message AuxillaryData {  

    string name = 1;  

    Encoding encoding = 2;  

    oneof data {  

        bytes byte_data = 3;  

        SharedMemoryHandle shared_memory_handle = 4;  

    }  

}  

//  

// Tensor represents a tensor, encoded as contiguous multi-dimensional array.  

// tensor_metadata - represents metadata of the shared memory segment  

// data_or_handle - represents the data of shared memory, this could be passed in two  

// ways:

```

```
// a. send across the raw bytes of the multi-dimensional tensor array
// b. send a SharedMemoryHandle which contains the posix shared memory segment
// id and offset in bytes to location of multi-dimensional tensor array.
//
message Tensor {
    TensorMetadata tensor_metadata = 1; //optional in the predict request
    oneof data {
        bytes byte_data = 4;
        // will only be used for input tensors
        SharedMemoryHandle shared_memory_handle = 5;
    }
}

//
// request for CaptureData rpc call
//
message CaptureDataRequest {
    string model_name = 1;
    string capture_id = 2; //uuid string
    Timestamp inference_timestamp = 3;
    repeated Tensor input_tensors = 4;
    repeated Tensor output_tensors = 5;
    repeated AuxiliaryData inputs = 6;
    repeated AuxiliaryData outputs = 7;
}
```

#### Output

```
//
// response for CaptureData rpc call
//
message CaptureDataResponse {}
```

## Get Capture Status

Depending on the models loaded the input and output tensors can be large (for many edge devices). Capture to the cloud can be time consuming. So the `CaptureData()` is implemented as an asynchronous operation. A capture ID is a unique identifier that the client provides during capture data call, this ID can be used to query the status of the asynchronous call.

```
//
// allows users to query status of capture data operation
// Status Codes:
// 1. OK - data capture successfully initiated
// 2. UNKNOWN - unknown error has occurred
// 3. INTERNAL - an internal error has occurred
// 4. NOT_FOUND - given capture id doesn't exist.
//
rpc GetCaptureDataStatus(GetCaptureDataStatusRequest) returns
    (GetCaptureDataStatusResponse);
```

#### Input

```
//
// request for GetCaptureDataStatus rpc call
//
message GetCaptureDataStatusRequest {
    string capture_id = 1;
}
```

## Output

```

enum CaptureDataStatus {
    FAILURE = 0;
    SUCCESS = 1;
    IN_PROGRESS = 2;
    NOT_FOUND = 3;
}

//
// response for GetCaptureDataStatus rpc call
//
message GetCaptureDataStatusResponse {
    CaptureDataStatus status = 1;
}

```

## Predict

The predict API performs inference on a previously loaded model. It accepts a request in the form of a tensor that is directly fed into the neural network. The output is the output tensor (or scalar) from the model. This is a blocking call.

```

//
// perform inference on a model.
//
// Note:
// 1. users can chose to send the tensor data in the protobuf message or
// through a shared memory segment on a per tensor basis, the Predict
// method will handle the decode transparently.
// 2. serializing large tensors into the protobuf message can be quite expensive,
// based on our measurements it is recommended to use shared memory of
// tenors larger than 256KB.
// 3. SMEdge IPC server will not use shared memory for returning output tensors,
// i.e., the output tensor data will always send in byte form encoded
// in the tensors of PredictResponse.
// 4. currently SMEdge IPC server cannot handle concurrent predict calls, all
// these call will be serialized under the hood. this shall be addressed
// in a later release.
// Status Codes:
// 1. OK - prediction is successful
// 2. UNKNOWN - unknown error has occurred
// 3. INTERNAL - an internal error has occurred
// 4. NOT_FOUND - when model not found
// 5. INVALID_ARGUMENT - when tenors types mismatch
//
rpc Predict(PredictRequest) returns (PredictResponse);

```

## Input

```

// request for Predict rpc call
//
message PredictRequest {
    string name = 1;
    repeated Tensor tensors = 2;
}

//
// Tensor represents a tensor, encoded as contiguous multi-dimensional array.
//     tensor_metadata - represents metadata of the shared memory segment
//     data_or_handle - represents the data of shared memory, this could be passed in
// two ways:

```

```

//                                     a. send across the raw bytes of the multi-dimensional tensor
array
//                                     b. send a SharedMemoryHandle which contains the posix shared
memory segment
//                                         id and offset in bytes to location of multi-dimensional
tensor array.
//
message Tensor {
    TensorMetadata tensor_metadata = 1; //optional in the predict request
    oneof data {
        bytes byte_data = 4;
        // will only be used for input tensors
        SharedMemoryHandle shared_memory_handle = 5;
    }
}

//
// Tensor represents a tensor, encoded as contiguous multi-dimensional array.
//   tensor_metadata - represents metadata of the shared memory segment
//   data_or_handle - represents the data of shared memory, this could be passed in
two ways:
//                                     a. send across the raw bytes of the multi-dimensional tensor
array
//                                     b. send a SharedMemoryHandle which contains the posix shared
memory segment
//                                         id and offset in bytes to location of multi-dimensional
tensor array.
//
message Tensor {
    TensorMetadata tensor_metadata = 1; //optional in the predict request
    oneof data {
        bytes byte_data = 4;
        // will only be used for input tensors
        SharedMemoryHandle shared_memory_handle = 5;
    }
}

//
// TensorMetadata represents the metadata for a tensor
//   name - name of the tensor
//   data_type - data type of the tensor
//   shape - array of dimensions of the tensor
//
message TensorMetadata {
    string name = 1;
    DataType data_type = 2;
    repeated int32 shape = 3;
}

//
// SharedMemoryHandle represents a posix shared memory segment
//   offset - offset in bytes from the start of the shared memory segment.
//   segment_id - shared memory segment id corresponding to the posix shared memory
segment.
//   size - size in bytes of shared memory segment to use from the offset position.
//
message SharedMemoryHandle {
    uint64 size = 1;
    uint64 offset = 2;
    uint64 segment_id = 3;
}

```

## Output

### Note

The `PredictResponse` only returns `Tensors` and not `SharedMemoryHandle`.

```
// response for Predict rpc call
//
message PredictResponse {
    repeated Tensor tensors = 1;
}
```

# Using Docker containers with SageMaker

Amazon SageMaker makes extensive use of *Docker containers* for build and runtime tasks. SageMaker provides prebuilt Docker images for its built-in algorithms and the supported deep learning frameworks used for training and inference. Using containers, you can train machine learning algorithms and deploy models quickly and reliably at any scale. The topics in this section show how to deploy these containers for your own use cases.

## Topics

- [Scenarios for Running Scripts, Training Algorithms, or Deploying Models with SageMaker \(p. 2134\)](#)
- [Docker Container Basics \(p. 2135\)](#)
- [Use Prebuilt SageMaker Docker images \(p. 2135\)](#)
- [Adapting Your Own Docker Container to Work with SageMaker \(p. 2148\)](#)
- [Create a container with your own algorithms and models \(p. 2160\)](#)
- [Example Notebooks: Use Your Own Algorithm or Model \(p. 2175\)](#)
- [Troubleshooting your Docker containers \(p. 2177\)](#)

## Scenarios for Running Scripts, Training Algorithms, or Deploying Models with SageMaker

Amazon SageMaker always uses Docker containers when running scripts, training algorithms, and deploying models. However, your level of engagement with containers depends on your use case.

- **Use a built-in SageMaker algorithm or framework.** For most use cases, you can use the built-in algorithms and frameworks without worrying about containers. You can train and deploy these algorithms from the SageMaker console, the Amazon Command Line Interface (Amazon CLI), a Python notebook, or the [Amazon SageMaker Python SDK](#) by specifying the algorithm or framework version when creating your Estimator. The built-in algorithms available are itemized and described in the [Use Amazon SageMaker Built-in Algorithms \(p. 718\)](#) topic. For more information on the available frameworks, see [ML Frameworks and Toolkits \(p. 16\)](#). For an example of how to train and deploy a built-in algorithm using a Jupyter notebook running in a SageMaker notebook instance, see the [Get Started with Amazon SageMaker \(p. 34\)](#) topic.
- **Use prebuilt SageMaker container images.** Alternatively, you can use the built-in algorithms and frameworks using Docker containers. SageMaker provides containers for its built-in algorithms and prebuilt Docker images for some of the most common machine learning frameworks, such as Apache MXNet, TensorFlow, PyTorch, and Chainer. For a full list of the available SageMaker Images, see [Available Deep Learning Containers Images](#). It also supports machine learning libraries such as scikit-learn and SparkML. If you use the [Amazon SageMaker Python SDK](#), you can deploy the containers by passing the full container URI to their respective SageMaker SDK Estimator class. For the full list of deep learning frameworks currently supported by SageMaker, see [Prebuilt SageMaker Docker Images for Deep Learning \(p. 2135\)](#). For information on the scikit-learn and SparkML prebuilt container images, see [Prebuilt Amazon SageMaker Docker Images for Scikit-learn and Spark ML \(p. 2136\)](#). For more information about using frameworks with the [Amazon SageMaker Python SDK](#), see their respective topics in [Use Machine Learning Frameworks, Python, and R with Amazon SageMaker \(p. 16\)](#).
- **Extend a prebuilt SageMaker container image.** If you would like to extend a prebuilt SageMaker algorithm or model Docker image, you can modify the SageMaker image to satisfy your needs. For an example, see [Extending our PyTorch containers](#).

- **Adapt an existing container image:** If you would like to adapt a pre-existing container image to work with SageMaker, you need to modify the Docker container to enable either the SageMaker Training or Inference toolkit. For an example that shows how to build your own containers to train and host an algorithm, see [Bring Your Own R Algorithm](#).

## Docker Container Basics

Docker is a program that performs operating system-level virtualization for installing, distributing, and managing software. It packages applications and their dependencies into virtual containers that provide isolation, portability, and security. With Docker, you can ship code faster, standardize application operations, seamlessly move code, and economize by improving resource utilization. For more general information about Docker, see [Docker overview](#).

The following information outlines the most significant aspects of using Docker containers with Amazon SageMaker.

### SageMaker Functions

SageMaker uses Docker containers in the backend to manage training and inference processes. SageMaker abstracts away from this process, so it happens automatically when an estimator is used. While you don't need to use Docker containers explicitly with SageMaker for most use cases, you can use Docker containers to extend and customize SageMaker functionality.

### Containers with SageMaker Studio

SageMaker Studio runs from a Docker container and uses it to manage functionality. As a result, you cannot create and upload a Docker container from a SageMaker Studio instance. However, you can use a prebuilt SageMaker container as long as that container was created outside of Studio.

## Use Prebuilt SageMaker Docker images

Amazon SageMaker provides containers for its built-in algorithms and prebuilt Docker images for some of the most common machine learning frameworks, such as Apache MXNet, TensorFlow, PyTorch, and Chainer. It also supports machine learning libraries such as scikit-learn and SparkML.

You can use these images from your SageMaker notebook instance or SageMaker Studio. You can also extend the prebuilt SageMaker images to include libraries and needed functionality. The following topics give information about the available images and how to use them.

### Note

For information on Docker images for developing reinforcement learning (RL) solutions in SageMaker, see [SageMaker RL Containers](#).

### Topics

- [Prebuilt SageMaker Docker Images for Deep Learning \(p. 2135\)](#)
- [Prebuilt Amazon SageMaker Docker Images for Scikit-learn and Spark ML \(p. 2136\)](#)
- [Train a Deep Graph Network \(p. 2138\)](#)
- [Extend a Prebuilt Container \(p. 2140\)](#)

## Prebuilt SageMaker Docker Images for Deep Learning

SageMaker provides prebuilt Docker images that include deep learning framework libraries and other dependencies needed for training and inference. For a complete list of the available pre-built Docker images, see [Deep Learning Containers Images](#).

If you are not using the [Amazon SageMaker Python SDK](#) and one of its estimators to retrieve the pre-built images, you have to retrieve them yourself.

## Using the SageMaker Python SDK

With the [SageMaker Python SDK](#), you can train and deploy models using these popular deep learning frameworks. For instructions on installing and using the SDK, see [Amazon SageMaker Python SDK](#). The following table lists the available frameworks and instructions on how to use them with the [SageMaker Python SDK](#):

Framework	Instructions
TensorFlow	<a href="#">Using TensorFlow with the SageMaker Python SDK</a>
MXNet	<a href="#">Using MXNet with the SageMaker Python SDK</a>
PyTorch	<a href="#">Using PyTorch with the SageMaker Python SDK</a>
Chainer	<a href="#">Using Chainer with the SageMaker Python SDK</a>
Hugging Face	<a href="#">Using Hugging Face with the SageMaker Python SDK</a>

## Extending Prebuilt SageMaker Docker Images

You can customize these prebuilt containers or extend them to handle any additional functional requirements for your algorithm or model that the prebuilt SageMaker Docker image doesn't support. For an example, see [Extending Our PyTorch Containers](#).

You can also use prebuilt containers to deploy your custom models or models that have been trained in a framework other than SageMaker. For an overview of the process of bringing the trained model artifacts into SageMaker and hosting them at an endpoint, see [Bring Your Own Pretrained MXNet or TensorFlow Models into Amazon SageMaker](#).

## Prebuilt Amazon SageMaker Docker Images for Scikit-learn and Spark ML

SageMaker provides prebuilt Docker images that install the scikit-learn and Spark ML libraries. These libraries also include the dependencies needed to build Docker images that are compatible with SageMaker using the [Amazon SageMaker Python SDK](#). With the SDK, you can use scikit-learn for machine learning tasks and use Spark ML to create and tune machine learning pipelines. For instructions on installing and using the SDK, see [SageMaker Python SDK](#).

## Using the SageMaker Python SDK

The following table contains links to the GitHub repositories with the source code for the scikit-learn and Spark ML containers. The table also contains links to instructions that show how use these containers with Python SDK estimators to run your own training algorithms and hosting your own models.

Library	Prebuilt Docker Image Source Code	Instructions
scikit-learn	<a href="#">SageMaker Scikit-learn Containers</a>	<a href="#">Using Scikit-learn with the Amazon SageMaker Python SDK</a>
Spark ML	<a href="#">SageMaker Spark ML Serving Containers</a>	<a href="#">SparkML Python SDK Documentation</a>

## Specifying the Prebuilt Images Manually

If you are not using the SageMaker Python SDK and one of its estimators to manage the container, you have to retrieve the relevant prebuilt container manually. The SageMaker prebuilt Docker images are stored in Amazon Elastic Container Registry (Amazon ECR). You can push or pull them using their fullname registry addresses. SageMaker uses the following Docker Image URL patterns for scikit-learn and Spark M:

- <ACCOUNT\_ID>.dkr.ecr.<REGION\_NAME>.amazonaws.com/sagemaker-scikit-learn:<SCIKIT-LEARN\_VERSION>-cpu-py<PYTHON\_VERSION>

For example, `746614075791.dkr.ecr.us-west-1.amazonaws.com/sagemaker-scikit-learn:0.23-1-cpu-py3`

- <ACCOUNT\_ID>.dkr.ecr.<REGION\_NAME>.amazonaws.com/sagemaker-sparkml-serving:<SPARK-ML\_VERSION>

For example, `341280168497.dkr.ecr.ca-central-1.amazonaws.com/sagemaker-sparkml-serving:2.4`

The following table lists the supported values for account IDs and corresponding Amazon Region names.

ACCOUNT_ID	REGION_NAME
746614075791	us-west-1
246618743249	us-west-2
683313688378	us-east-1
257758044811	us-east-2
354813040037	ap-northeast-1
366743142698	ap-northeast-2
121021644041	ap-southeast-1
783357654285	ap-southeast-2
720646828776	ap-south-1
141502667606	eu-west-1
764974769150	eu-west-2
492215442770	eu-central-1
341280168497	ca-central-1
414596584902	us-gov-west-1

## Finding Available Images

Use the following commands to find out which versions of the images are available. For example, use the following to find the available sagemaker-sparkml-serving image in the ca-central-1 Region:

```
aws \
```

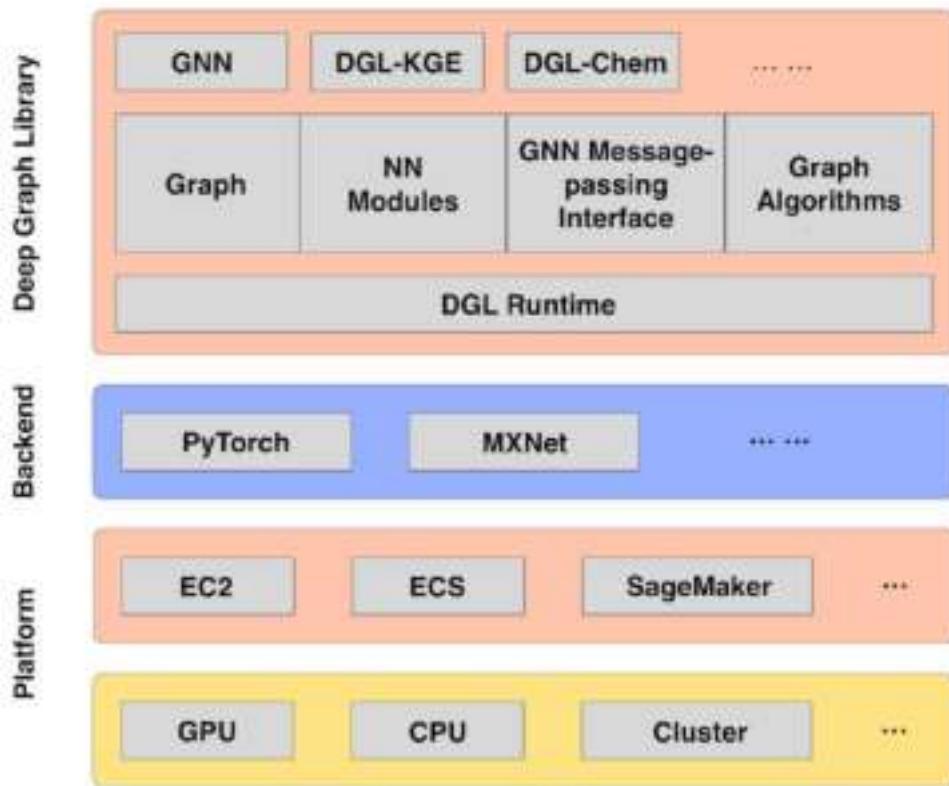
```
ecr describe-images \
--region ca-central-1 \
--repository-id 341280168497 \
--repository-name sagemaker-sparkml-serving
```

## Train a Deep Graph Network

In this overview, you learn how to get started with a deep graph network by using one of the DGL containers in Amazon Elastic Container Registry (Amazon ECR). You can also see links to practical examples for deep graph networks.

### What Is a Deep Graph Network?

Deep graph networks refer to a type of neural network that is trained to solve graph problems. A deep graph network uses an underlying deep learning framework like PyTorch or MXNet. The potential for graph networks in practical AI applications is highlighted in the Amazon SageMaker tutorials for [Deep Graph Library](#) (DGL). Examples for training models on graph datasets include social networks, knowledge bases, biology, and chemistry.



*Figure 1. The DGL ecosystem*

Several examples are provided using Amazon SageMaker's deep learning containers that are preconfigured with DGL. If you have special modules you want to use with DGL, you can also build your own container. The examples involve heterographs, which are graphs that have multiple types of nodes and edges, and draw on a variety of applications across disparate scientific fields, such as bioinformatics and social network analysis. DGL provides a wide array of [graph neural network implementations for different types models](#). Some of the highlights include:

- Graph convolutional network (GCN)
- Relational graph convolutional network (R-GCN)
- Graph attention network (GAT)
- Deep generative models of graphs (DGMG)
- Junction tree neural network (JTNN)

## Get Started

DGL is available as a deep learning container in Amazon ECR. You can select deep learning containers when you write your estimator function in an Amazon SageMaker notebook. You can also craft your own custom container with DGL by following the [Bring Your Own Container](#) guide. The easiest way to get started with a deep graph network uses one of the DGL containers in Amazon ECR.

### Note

Backend framework support is limited to PyTorch and MXNet.

### Setup

If you are using Amazon SageMaker Studio, you need to clone the examples repository first. If you are using a notebook instance, you can find the examples by choosing the SageMaker icon at bottom of the left toolbar.

### To clone the Amazon SageMaker SDK and notebook examples repository

1. From the **JupyterLab** view in Amazon SageMaker, go to the **File Browser** at the top of the left toolbar. From the **File Browser panel**, you can see a new navigation at the top of the panel.
2. Choose the icon on the far right to clone a Git repository.
3. Add the repository URL: <https://github.com/awslabs/amazon-sagemaker-examples.git>
4. Browse the newly added folder and its contents. The DGL examples are stored in the **sagemaker-python-sdk** folder.

## Run a Graph Network Training Example

### To train a deep graph network

1. From the **JupyterLab** view in Amazon SageMaker, browse the [example notebooks](#) and look for DGL folders. Several files may be included to support an example. Examine the README for any prerequisites.
2. Run the .ipynb notebook example.
3. Find the estimator function, and note the line where it is using an Amazon ECR container for DGL and a specific instance type. You may want to update this to use a container in your preferred Region.
4. Run the function to launch the instance and use the DGL container for training a graph network. Charges are incurred for launching this instance. The instance self-terminates when the training is complete.

## Examples

An example of knowledge graph embedding (KGE) is provided. It uses the Freebase dataset, a knowledge base of general facts. An example use case would be to graph the relationships of persons and predict their nationality.

An example implementation of a graph convolutional network (GCN) shows how you can train a graph network to predict toxicity. A physiology dataset, Tox21, provides toxicity measurements for how substances affect biological responses.

Another GCN example shows you how to train a graph network on a scientific publications bibliography dataset, known as Cora. You can use it to find relationships between authors, topics, and conferences.

The last example is a recommender system for movie reviews. It uses a graph convolutional matrix completion (GCMC) network trained on the MovieLens datasets. These datasets consist of movie titles, genres, and ratings by users.

## Use a Deep Learning Container with DGL

The following examples use preconfigured deep learning containers. These are the easiest to try since they work out of the box on Amazon SageMaker.

- [Semi-supervised classification of a knowledge base using a GCN](#)
- [Learning embeddings of large-scale knowledge graphs using a dataset of scientific publications](#)

## Bring Your Own Container with DGL

The following examples enable you to bring your own container (BYOC). Read the [BYOC guide](#) and familiarize yourself with that process before trying these. Configuration is required.

- [Molecular property prediction of toxicity using a GCN](#)
- [Recommender system for movies using a GCMC implementation](#)

# Extend a Prebuilt Container

If a prebuilt SageMaker container doesn't fulfill all of your requirements, you can extend the existing image to accommodate your needs. Even if there is direct support for your environment or framework, you may want to add additional functionality or configure your container environment differently. By extending a prebuilt image, you can leverage the included deep learning libraries and settings without having to create an image from scratch. You can extend the container to add libraries, modify settings, and install additional dependencies.

The following tutorial shows how to extend a prebuilt SageMaker image and publish it to Amazon ECR.

### Topics

- [Requirements to Extend a Prebuilt Container \(p. 2140\)](#)
- [Extend SageMaker Containers to Run a Python Script \(p. 2141\)](#)

## Requirements to Extend a Prebuilt Container

To extend a pre-built SageMaker image, you need to set the following environment variables within your Dockerfile. For more information on environment variables with SageMaker containers, see the [SageMaker Training Toolkit GitHub repo](#).

- **SAGEMAKER\_SUBMIT\_DIRECTORY:** The directory within the container in which the Python script for training is located.
- **SAGEMAKER\_PROGRAM:** The Python script that should be invoked and used as the entry point for training.

You can also install additional libraries by including the following in your Dockerfile:

```
RUN pip install <library>
```

The following tutorial shows how to use these environment variables.

## Extend SageMaker Containers to Run a Python Script

In this tutorial, you learn how to extend the SageMaker PyTorch container with a Python file that uses the CIFAR-10 dataset. By extending the SageMaker PyTorch container, you utilize the existing training solution made to work with SageMaker. This tutorial extends a training image, but the same steps can be taken to extend an inference image. For a full list of the available images, see [Available Deep Learning Containers Images](#).

To run your own training model using the SageMaker containers, build a Docker container through a SageMaker Notebook instance.

### Step 1: Create an SageMaker Notebook Instance

1. Open the [SageMaker console](#).
2. In the left navigation pane, choose **Notebook**, choose **Notebook instances**, and then choose **Create notebook instance**.
3. On the **Create notebook instance** page, provide the following information:
  - a. For **Notebook instance name**, enter **RunScriptNotebookInstance**.
  - b. For **Notebook Instance type**, choose **m1.t2.medium**.
  - c. In the **Permissions and encryption** section, do the following:
    - i. For **IAM role**, choose **Create a new role**.
    - ii. On the **Create an IAM role** page, choose **Specific S3 buckets**, specify an Amazon S3 bucket named **sagemaker-run-script**, and then choose **Create role**.SageMaker creates an IAM role named **AmazonSageMaker-ExecutionRole-YYYYMMDDTHHmSS**, such as **AmazonSageMaker-ExecutionRole-20190429T110788**. Note that the execution role naming convention uses the date and time when the role was created, separated by a **T**.
  - d. For **Root Access**, choose **Enable**.
  - e. Choose **Create notebook instance**.
4. On the **Notebook instances** page, the **Status** is **Pending**. It can take a few minutes for Amazon SageMaker to launch a machine learning compute instance—in this case, it launches a notebook instance—and attach an ML storage volume to it. The notebook instance has a preconfigured Jupyter notebook server and a set of Anaconda libraries. For more information, see [CreateNotebookInstance](#).
5. In the **Permissions and encryption** section, copy the **IAM role ARN number**, and paste it into a notepad file to save it temporarily. You use this IAM role ARN number later to configure a local training estimator in the notebook instance. The **IAM role ARN number** looks like the following: `'arn:aws:iam::111122223333:role/service-role/AmazonSageMaker-ExecutionRole-20190429T110788'`
6. After the status of the notebook instance changes to **InService**, choose **Open JupyterLab**.

### Step 2: Create and Upload the Dockerfile and Python Training Scripts

1. After JupyterLab opens, create a new folder in the home directory of your JupyterLab. In the upper-left corner, choose the **New Folder** icon, and then enter the folder name `docker_test_folder`.

2. Create a Dockerfile text file in the docker\_test\_folder directory.
  - a. Choose the **New Launcher** icon (+) in the upper-left corner.
  - b. In the right pane under the **Other** section, choose **Text File**.
  - c. Paste the following Dockerfile sample code into your text file.

```
# SageMaker PyTorch image
FROM 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-training:1.5.1-cpu-py36-
ubuntu16.04

ENV PATH="/opt/ml/code:${PATH}"

# this environment variable is used by the SageMaker PyTorch container to determine
# our user code directory.
ENV SAGEMAKER_SUBMIT_DIRECTORY /opt/ml/code

# /opt/ml and all subdirectories are utilized by SageMaker, use the /code
# subdirectory to store your user code.
COPY cifar10.py /opt/ml/code/cifar10.py

# Defines cifar10.py as script entrypoint
ENV SAGEMAKER_PROGRAM cifar10.py
```

The Dockerfile script performs the following tasks:

- FROM 763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-training:1.5.1-cpu-py36-ubuntu16.04 – Downloads the SageMaker PyTorch base image. You can replace this with any SageMaker base image you want to bring to build containers.
  - ENV SAGEMAKER\_SUBMIT\_DIRECTORY /opt/ml/code – Sets /opt/ml/code as the training script directory.
  - COPY cifar10.py /opt/ml/code/cifar10.py – Copies the script to the location inside the container that is expected by SageMaker. The script must be located in this folder.
  - ENV SAGEMAKER\_PROGRAM cifar10.py – Sets your cifar10.py training script as the entrypoint script.
- d. On the left directory navigation pane, the text file name might automatically be named `untitled.txt`. To rename the file, right-click the file, choose **Rename**, rename the file as `Dockerfile` without the `.txt` extension, and then press `Ctrl+s` or `Command+s` to save the file.
  3. Create or upload a training script `cifar10.py` in the `docker_test_folder`. You can use the following example script for this exercise.

```
import ast
import argparse
import logging

import os

import torch
import torch.distributed as dist
import torch.nn as nn
import torch.nn.parallel
import torch.optim
import torch.utils.data
import torch.utils.data.distributed
import torchvision
import torchvision.models
import torchvision.transforms as transforms
import torch.nn.functional as F
```

```

logger=logging.getLogger(__name__)
logger.setLevel(logging.DEBUG)

classes=('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship',
        'truck')

# https://github.com/pytorch/tutorials/blob/master/beginner_source/blitz/
cifar10_tutorial.py#L118
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1=nn.Conv2d(3, 6, 5)
        self.pool=nn.MaxPool2d(2, 2)
        self.conv2=nn.Conv2d(6, 16, 5)
        self.fc1=nn.Linear(16 * 5 * 5, 120)
        self.fc2=nn.Linear(120, 84)
        self.fc3=nn.Linear(84, 10)

    def forward(self, x):
        x=self.pool(F.relu(self.conv1(x)))
        x=self.pool(F.relu(self.conv2(x)))
        x=x.view(-1, 16 * 5 * 5)
        x=F.relu(self.fc1(x))
        x=F.relu(self.fc2(x))
        x=self.fc3(x)
        return x

def _train(args):
    is_distributed=len(args.hosts) > 1 and args.dist_backend is not None
    logger.debug("Distributed training - {}".format(is_distributed))

    if is_distributed:
        # Initialize the distributed environment.
        world_size=len(args.hosts)
        os.environ['WORLD_SIZE']=str(world_size)
        host_rank=args.hosts.index(args.current_host)
        dist.init_process_group(backend=args.dist_backend, rank=host_rank,
        world_size=world_size)
        logger.info(
            'Initialized the distributed environment: \'{}\' backend on {} nodes.
'.format(
            args.dist_backend,
            dist.get_world_size() + 'Current host rank is {}'.format(host_rank).Using cuda: {}'.format(
                dist.get_rank(), torch.cuda.is_available(), args.num_gpus))

        device='cuda' if torch.cuda.is_available() else 'cpu'
        logger.info("Device Type: {}".format(device))

        logger.info("Loading Cifar10 dataset")
        transform=transforms.Compose(
            [transforms.ToTensor(),
             transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

        trainset=torchvision.datasets.CIFAR10(root=args.data_dir, train=True,
                                              download=False, transform=transform)
        train_loader=torch.utils.data.DataLoader(trainset, batch_size=args.batch_size,
                                                shuffle=True, num_workers=args.workers)

        testset=torchvision.datasets.CIFAR10(root=args.data_dir, train=False,
                                             download=False, transform=transform)
        test_loader=torch.utils.data.DataLoader(testset, batch_size=args.batch_size,
                                                shuffle=False, num_workers=args.workers)

```

```

logger.info("Model loaded")
model=Net()

if torch.cuda.device_count() > 1:
    logger.info("Gpu count: {}".format(torch.cuda.device_count()))
    model=nn.DataParallel(model)

model=model.to(device)

criterion=nn.CrossEntropyLoss().to(device)
optimizer=torch.optim.SGD(model.parameters(), lr=args.lr, momentum=args.momentum)

for epoch in range(0, args.epochs):
    running_loss=0.0
    for i, data in enumerate(train_loader):
        # get the inputs
        inputs, labels=data
        inputs, labels=inputs.to(device), labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs=model(inputs)
        loss=criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999: # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            running_loss=0.0
    print('Finished Training')
    return _save_model(model, args.model_dir)

def _save_model(model, model_dir):
    logger.info("Saving the model.")
    path=os.path.join(model_dir, 'model.pth')
    # recommended way from http://pytorch.org/docs/master/notes/serialization.html
    torch.save(model.cpu().state_dict(), path)

def model_fn(model_dir):
    logger.info('model_fn')
    device="cuda" if torch.cuda.is_available() else "cpu"
    model=Net()
    if torch.cuda.device_count() > 1:
        logger.info("Gpu count: {}".format(torch.cuda.device_count()))
        model=nn.DataParallel(model)

    with open(os.path.join(model_dir, 'model.pth'), 'rb') as f:
        model.load_state_dict(torch.load(f))
    return model.to(device)

if __name__ == '__main__':
    parser=argparse.ArgumentParser()

    parser.add_argument('--workers', type=int, default=2, metavar='W',
                       help='number of data loading workers (default: 2)')
    parser.add_argument('--epochs', type=int, default=2, metavar='E',
                       help='number of total epochs to run (default: 2)')


```

```

parser.add_argument('--batch-size', type=int, default=4, metavar='BS',
                    help='batch size (default: 4)')
parser.add_argument('--lr', type=float, default=0.001, metavar='LR',
                    help='initial learning rate (default: 0.001)')
parser.add_argument('--momentum', type=float, default=0.9, metavar='M',
                    help='momentum (default: 0.9)')
parser.add_argument('--dist-backend', type=str, default='gloo', help='distributed
backend (default: gloo)')

# The parameters below retrieve their default values from SageMaker environment
variables, which are
# instantiated by the SageMaker containers framework.
# https://github.com/aws/sagemaker-containers#how-a-script-is-executed-inside-the-
container
parser.add_argument('--hosts', type=str,
default=ast.literal_eval(os.environ['SM_HOSTS']))
parser.add_argument('--current-host', type=str,
default=os.environ['SM_CURRENT_HOST'])
parser.add_argument('--model-dir', type=str, default=os.environ['SM_MODEL_DIR'])
parser.add_argument('--data-dir', type=str,
default=os.environ['SM_CHANNEL_TRAINING'])
parser.add_argument('--num-gpus', type=int, default=os.environ['SM_NUM_GPUS'])

_train(parser.parse_args())

```

## Step 3: Build the Container

1. In the JupyterLab home directory, open a Jupyter notebook. To open a new notebook, choose the **New Launch** icon and then choose **conda\_pytorch\_p36** in the **Notebook** section.
2. Run the following command in the first notebook cell to change to the `docker_test_folder` directory:

```
% cd ~/SageMaker/docker_test_folder
```

This returns your current directory as follows:

```
! pwd
```

```
output: /home/ec2-user/SageMaker/docker_test_folder
```

3. Log in to Docker to access the base container:

```
! aws ecr get-login-password --region us-east-1 | docker login --username AWS --
password-stdin 763104351884.dkr.ecr.us-east-1.amazonaws.com
```

4. To build the Docker container, run the following Docker build command, including the space followed by a period at the end:

```
! docker build -t pytorch-extended-container-test .
```

The Docker build command must be run from the Docker directory you created, in this case `docker_test_folder`.

**Note**

If you get the following error message that Docker cannot find the Dockerfile, make sure the Dockerfile has the correct name and has been saved to the directory.

```
unable to prepare context: unable to evaluate symlinks in Dockerfile path:
```

```
lstat /home/ec2-user/SageMaker/docker/Dockerfile: no such file or directory
```

Remember that docker looks for a file specifically called Dockerfile without any extension within the current directory. If you named it something else, you can pass in the file name manually with the -f flag. For example, if you named your Dockerfile Dockerfile-text.txt, run the following command:

```
! docker build -t tf-custom-container-test -f Dockerfile-text.txt .
```

## Step 4: Test the Container

1. To test the container locally in the notebook instance, open a Jupyter notebook. Choose **New Launcher** and choose **Notebook** in **conda\_pytorch\_p36** framework. The rest of the code snippets must run from the Jupyter notebook instance.
2. Download the CIFAR-10 dataset.

```
import torch
import torchvision
import torchvision.transforms as transforms

def _get_transform():
    return transforms.Compose(
        [transforms.ToTensor(),
         transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

def get_train_data_loader(data_dir='/tmp/pytorch/cifar-10-data'):
    transform=_get_transform()
    trainset=torchvision.datasets.CIFAR10(root=data_dir, train=True,
                                           download=True, transform=transform)
    return torch.utils.data.DataLoader(trainset, batch_size=4,
                                       shuffle=True, num_workers=2)

def get_test_data_loader(data_dir='/tmp/pytorch/cifar-10-data'):
    transform=_get_transform()
    testset=torchvision.datasets.CIFAR10(root=data_dir, train=False,
                                         download=True, transform=transform)
    return torch.utils.data.DataLoader(testset, batch_size=4,
                                       shuffle=False, num_workers=2)

trainloader=get_train_data_loader('/tmp/pytorch-example/cifar-10-data')
testloader=get_test_data_loader('/tmp/pytorch-example/cifar-10-data')
```

3. Set role to the role used to create your Jupyter notebook. This is used to configure your SageMaker Estimator.

```
from sagemaker import get_execution_role
role=get_execution_role()
```

4. Paste the following example script into the notebook code cell to configure a SageMaker Estimator using your extended container.

```
from sagemaker.estimator import Estimator
hyperparameters={'epochs': 1}
estimator=Estimator(
```

```

        image_uri='pytorch-extended-container-test',
        role=role,
        instance_count=1,
        instance_type='local',
        hyperparameters=hyperparameters
    )

estimator.fit('file:///tmp/pytorch-example/cifar-10-data')

```

- Run the code cell. This test outputs the training environment configuration, the values used for the environmental variables, the source of the data, and the loss and accuracy obtained during training.

## Step 5: Push the Container to Amazon Elastic Container Registry (Amazon ECR)

- After you successfully run the local mode test, you can push the Docker container to [Amazon ECR](#) and use it to run training jobs.

Run the following command lines in a notebook cell.

```

%%sh

# Specify an algorithm name
algorithm_name=pytorch-extended-container-test

account=$(aws sts get-caller-identity --query Account --output text)

# Get the region defined in the current configuration (default to us-west-2 if none
# defined)
region=$(aws configure get region)

fullname="${account}.dkr.ecr.${region}.amazonaws.com/${algorithm_name}:latest"

# If the repository doesn't exist in ECR, create it.

aws ecr describe-repositories --repository-names "${algorithm_name}" > /dev/null 2>&1
if [ $? -ne 0 ]
then
aws ecr create-repository --repository-name "${algorithm_name}" > /dev/null
fi

# Build the docker image locally with the image name and then push it to ECR
# with the full name.

docker build -t ${algorithm_name} .
docker tag ${algorithm_name} ${fullname}

docker push ${fullname}

```

- After you push the container, you can call the Amazon ECR image from anywhere in the SageMaker environment. Run the following code example in the next notebook cell.

If you want to use this training container with SageMaker Studio to use its visualization features, you can also run the following code in a Studio notebook cell to call the Amazon ECR image of your training container.

```

import boto3

client=boto3.client('sts')
account=client.get_caller_identity()['Account']

my_session=boto3.session.Session()
region=my_session.region_name

```

```
algorithm_name="pytorch-extended-container-test"
ecr_image='{}.dkr.ecr.{}.amazonaws.com/{}:latest'.format(account, region,
algorithm_name)

ecr_image
# This should return something like
# 12-digits-of-your-account.dkr.ecr.us-east-2.amazonaws.com/tf-2.2-test:latest
```

3. Use the `ecr_image` retrieved from the previous step to configure a SageMaker estimator object. The following code sample configures a SageMaker PyTorch estimator.

```
import sagemaker

from sagemaker import get_execution_role
from sagemaker.estimator import Estimator

estimator=Estimator(
    image_uri=ecr_image,
    role=get_execution_role(),
    base_job_name='pytorch-extended-container-test',
    instance_count=1,
    instance_type='ml.p2.xlarge'
)

# start training
estimator.fit()

# deploy the trained model
predictor=estimator.deploy(1, instance_type)
```

## Step 6: Clean up Resources

### To clean up resources when done with the Get Started example

1. Open the [SageMaker console](#), choose the notebook instance **RunScriptNotebookInstance**, choose **Actions**, and choose **Stop**. It can take a few minutes for the instance to stop.
2. After the instance **Status** changes to **Stopped**, choose **Actions**, choose **Delete**, and then choose **Delete** in the dialog box. It can take a few minutes for the instance to be deleted. The notebook instance disappears from the table when it has been deleted.
3. Open the [Amazon S3 console](#) and delete the bucket that you created for storing model artifacts and the training dataset.
4. Open the [IAM console](#) and delete the IAM role. If you created permission policies, you can delete them, too.

#### Note

The Docker container shuts down automatically after it has run. You don't need to delete it.

# Adapting Your Own Docker Container to Work with SageMaker

You can adapt an existing Docker image to work with SageMaker. You may need to use an existing, external Docker image with SageMaker when you have a container that satisfies feature or safety requirements that are not currently supported by a prebuilt SageMaker image. There are two toolkits that allow you to bring your own container and adapt it to work with SageMaker:

- [SageMaker Training Toolkit](#)
- [SageMaker Inference Toolkit](#)

The following topics show how to adapt your existing image using the SageMaker Training and Inference toolkits:

#### Topics

- [Individual Framework Libraries \(p. 2149\)](#)
- [Using the SageMaker Training and Inference Toolkits \(p. 2149\)](#)
- [Adapting Your Own Training Container \(p. 2151\)](#)
- [Adapting Your Own Inference Container \(p. 2156\)](#)

## Individual Framework Libraries

In addition to the SageMaker Training Toolkit and SageMaker Inference Toolkit, SageMaker also provides toolkits specialized for TensorFlow, MXNet, PyTorch, and Chainer. The following table provides links to the GitHub repositories that contain the source code for each framework and their respective serving toolkits. The instructions linked are for using the Python SDK to run training algorithms and host models on SageMaker. The functionality for these individual libraries is included in the SageMaker Training Toolkit and SageMaker Inference Toolkit.

Framework	Toolkit Source Code
TensorFlow	<a href="#">SageMaker TensorFlow Training</a> <a href="#">SageMaker TensorFlow Serving</a>
MXNet	<a href="#">SageMaker MXNet Training</a> <a href="#">SageMaker MXNet Inference</a>
PyTorch	<a href="#">SageMaker PyTorch Training</a> <a href="#">SageMaker PyTorch Inference</a>
Chainer	<a href="#">SageMaker Chainer SageMaker Containers</a>

## Using the SageMaker Training and Inference Toolkits

The [SageMaker Training](#) and [SageMaker Inference](#) toolkits implement the functionality that you need to adapt your containers to run scripts, train algorithms, and deploy models on SageMaker. When installed, the library defines the following for users:

- The locations for storing code and other resources.
- The entry point that contains the code to run when the container is started. Your Dockerfile must copy the code that needs to be run into the location expected by a container that is compatible with SageMaker.
- Other information that a container needs to manage deployments for training and inference.

## SageMaker Toolkits Containers Structure

When SageMaker trains a model, it creates the following file folder structure in the container's `/opt/ml` directory.

```
/opt/ml
### input
#   ### config
#   #   ### hyperparameters.json
#   #   ### resourceConfig.json
#   ### data
#       ### <channel_name>
#           ### <input data>
### model
#
### code
#
### output
#
### failure
```

When you run a model *training* job, the SageMaker container uses the `/opt/ml/input/` directory, which contains the JSON files that configure the hyperparameters for the algorithm and the network layout used for distributed training. The `/opt/ml/input/` directory also contains files that specify the channels through which SageMaker accesses the data, which is stored in Amazon Simple Storage Service (Amazon S3). The SageMaker containers library places the scripts that the container will run in the `/opt/ml/code/` directory. Your script should write the model generated by your algorithm to the `/opt/ml/model/` directory. For more information, see [Use Your Own Training Algorithms \(p. 2160\)](#).

When you *host* a trained model on SageMaker to make inferences, you deploy the model to an HTTP endpoint. The model makes real-time predictions in response to inference requests. The container must contain a serving stack to process these requests.

In a hosting or batch transform container, the model files are located in the same folder to which they were written during training.

```
/opt/ml/model
#
### <model files>
```

For more information, see [Use Your Own Inference Code \(p. 2166\)](#).

## Single Versus Multiple Containers

You can either provide separate Docker images for the training algorithm and inference code or you can use a single Docker image for both. When creating Docker images for use with SageMaker, consider the following:

- Providing two Docker images can increase storage requirements and cost because common libraries might be duplicated.
- In general, smaller containers start faster for both training and hosting. Models train faster and the hosting service can react to increases in traffic by automatically scaling more quickly.
- You might be able to write an inference container that is significantly smaller than the training container. This is especially common when you use GPUs for training, but your inference code is optimized for CPUs.
- SageMaker requires that Docker containers run without privileged access.

- Both Docker containers that you build and those provided by SageMaker can send messages to the `Stdout` and `Stderr` files. SageMaker sends these messages to Amazon CloudWatch logs in your Amazon account.

For more information about how to create SageMaker containers and how scripts are executed inside them, see the [SageMaker Training Toolkit](#) and [SageMaker Inference Toolkit](#) repositories on GitHub. They also provide lists of important environmental variables and the environmental variables provided by SageMaker containers.

## Adapting Your Own Training Container

To run your own training model, build a Docker container using the [Amazon SageMaker Training Toolkit](#) through an Amazon SageMaker notebook instance.

### Step 1: Create a SageMaker notebook instance

- Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
- In the left navigation pane, choose **Notebook**, choose **Notebook instances**, and then choose **Create notebook instance**.
- On the **Create notebook instance** page, provide the following information:
  - For **Notebook instance name**, enter `RunScriptNotebookInstance`.
  - For **Notebook Instance type**, choose `m1.t2.medium`.
  - In the **Permissions and encryption** section, do the following:
    - For **IAM role**, choose **Create a new role**.
    - On the **Create an IAM role** page, choose **Specific S3 buckets**, specify an Amazon S3 bucket named `sagemaker-run-script`, and then choose **Create role**.

SageMaker creates an IAM role named `AmazonSageMaker-ExecutionRole-YYYYMMDDTHHmmSS`. For example, `AmazonSageMaker-ExecutionRole-20190429T110788`. Note that the execution role naming convention uses the date and time at which the role was created, separated by a `T`.
  - For **Root Access**, choose **Enable**.
  - Choose **Create notebook instance**.
- On the **Notebook instances** page, the **Status** is **Pending**. It can take a few minutes for Amazon SageMaker to launch a machine learning compute instance—in this case, it launches a notebook instance—and attach an ML storage volume to it. The notebook instance has a preconfigured Jupyter notebook server and a set of Anaconda libraries. For more information, see [CreateNotebookInstance](#).
- In the **Permissions and encryption** section, copy the **IAM role ARN number**, and paste it into a notepad file to save it temporarily. You use this IAM role ARN number later to configure a local training estimator in the notebook instance. The **IAM role ARN number** looks like the following: '`arn:aws:iam::111122223333:role/service-role/AmazonSageMaker-ExecutionRole-20190429T110788`'
- After the status of the notebook instance changes to **InService**, choose **Open JupyterLab**.

## Step 2: Create and upload the Dockerfile and Python training scripts

1. After JupyterLab opens, create a new folder in the home directory of your JupyterLab. In the upper-left corner, choose the **New Folder** icon, and then enter the folder name `docker_test_folder`.
2. Create a Dockerfile text file in the `docker_test_folder` directory.
  - a. Choose the **New Launcher** icon (+) in the upper-left corner.
  - b. In the right pane under the **Other** section, choose **Text File**.
  - c. Paste the following Dockerfile sample code into your text file.

```
FROM tensorflow/tensorflow:2.2.0rc2-gpu-py3-jupyter

# Install sagemaker-training toolkit that contains the common functionality
# necessary to create a container compatible with SageMaker and the Python SDK.
RUN pip3 install sagemaker-training

# Copies the training code inside the container
COPY train.py /opt/ml/code/train.py

# Defines train.py as script entrypoint
ENV SAGEMAKER_PROGRAM train.py
```

The Dockerfile script performs the following tasks:

- `FROM tensorflow/tensorflow:2.2.0rc2-gpu-py3-jupyter` – Downloads the TensorFlow Docker base image. You can replace this with any Docker base image you want to bring to build containers, as well as with Amazon pre-built container base images.
  - `RUN pip install sagemaker-training` – Installs [SageMaker Training Toolkit](#) that contains the common functionality necessary to create a container compatible with SageMaker.
  - `COPY train.py /opt/ml/code/train.py` – Copies the script to the location inside the container that is expected by SageMaker. The script must be located in this folder.
  - `ENV SAGEMAKER_PROGRAM train.py` – Takes your training script `train.py` as the entrypoint script copied in the `/opt/ml/code` folder of the container. This is the only environmental variable that you must specify when you build your own container.
- d. On the left directory navigation pane, the text file name might automatically be named `untitled.txt`. To rename the file, right-click the file, choose **Rename**, rename the file as `Dockerfile` without the `.txt` extension, and then press `Ctrl+s` or `Command+s` to save the file.
  3. Create or upload a training script `train.py` in the `docker_test_folder`. You can use the following example script for this exercise.

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

```
model.compile(optimizer='adam',
  loss='sparse_categorical_crossentropy',
  metrics=['accuracy'])

model.fit(x_train, y_train, epochs=1)

model.evaluate(x_test, y_test)
```

## Step 3: Build the container

1. In the JupyterLab home directory, open a Jupyter notebook. To open a new notebook, choose the **New Launch** icon and then choose **conda\_tensorflow2\_p36** in the **Notebook** section.
2. Run the following command in the first notebook cell to change to the `docker_test_folder` directory:

```
% cd ~/SageMaker/docker_test_folder
```

This returns your current directory as follows:

```
! pwd
```

```
output: /home/ec2-user/SageMaker/docker_test_folder
```

3. To build the Docker container, run the following Docker build command, including the space followed by a period at the end:

```
! docker build -t tf-custom-container-test .
```

The Docker build command must be run from the Docker directory you created, in this case `docker_test_folder`.

**Note**

If you get the following error message that Docker cannot find the Dockerfile, make sure the Dockerfile has the correct name and has been saved to the directory.

```
unable to prepare context: unable to evaluate symlinks in Dockerfile path:
lstat /home/ec2-user/SageMaker/docker/Dockerfile: no such file or directory
```

Remember that `docker` looks for a file specifically called `Dockerfile` without any extension within the current directory. If you named it something else, you can pass in the file name manually with the `-f` flag. For example, if you named your Dockerfile as `Dockerfile-text.txt`, run the following command:

```
! docker build -t tf-custom-container-test -f Dockerfile-text.txt .
```

## Step 4: Test the container

1. To test the container locally in the notebook instance, open a Jupyter notebook. Choose **New Launcher** and choose **Notebook** in **conda\_tensorflow\_p36** framework.
2. Paste the following example script into the notebook code cell to configure a SageMaker Estimator.

### SageMaker Python SDK v1

```
from sagemaker.estimator import Estimator

estimator = Estimator(image_name='tf-custom-container-test',
                      role='arn:aws:iam::111122223333:role/role-name',
                      instance_count=1,
                      instance_type='local')

estimator.fit()
```

### SageMaker Python SDK v2

```
from sagemaker.estimator import Estimator

estimator = Estimator(image_uri='tf-custom-container-test',
                      role='arn:aws:iam::111122223333:role/role-name',
                      instance_count=1,
                      instance_type='local')

estimator.fit()
```

3. Replace the 'Put\_Your\_ARN\_Here' value with **the IAM role ARN number** you copied to a notepad file when you configured the notebook instance. The ARN should look like the following: 'arn:aws:iam::111122223333:role/service-role/AmazonSageMaker-ExecutionRole-20190429T110788'.
4. Run the code cell. This test outputs the training environment configuration, the values used for the environmental variables, the source of the data, and the loss and accuracy obtained during training.

## Step 5: Push the container to Amazon Elastic Container Registry (Amazon ECR)

1. After you successfully run the local mode test, you can push the Docker container to [Amazon ECR](#) and use it to run training jobs.

Run the following command lines in a notebook cell.

```
%%sh

# Specify an algorithm name
algorithm_name=tf-custom-container-test

account=$(aws sts get-caller-identity --query Account --output text)

# Get the region defined in the current configuration (default to us-west-2 if none
# defined)
region=$(aws configure get region)
region=${region:-us-west-2}

fullname="${account}.dkr.ecr.${region}.amazonaws.com/${algorithm_name}:latest"

# If the repository doesn't exist in ECR, create it.

aws ecr describe-repositories --repository-names "${algorithm_name}" > /dev/null 2>&1
if [ $? -ne 0 ]
then
aws ecr create-repository --repository-name "${algorithm_name}" > /dev/null
fi
```

```
# Get the login command from ECR and execute it directly
$(aws ecr get-login --region ${region} --no-include-email)

# Build the docker image locally with the image name and then push it to ECR
# with the full name.

docker build -t ${algorithm_name} .
docker tag ${algorithm_name} ${fullname}

docker push ${fullname}
```

### Note

This bash shell script may raise a permission issue similar to the following error message:

```
"denied: User: [ARN] is not authorized to perform: ecr:InitiateLayerUpload on
resource:
arn:aws:ecr:us-east-1:[id]:repository/tf-custom-container-test"
```

If this error occurs, you need to attach the **AmazonEC2ContainerRegistryFullAccess** policy to your IAM role. Go to the [IAM console](#), choose **Roles** from the left navigation pane, look up the IAM role you used for the Notebook instance. Under the **Permission** tab, choose the **Attach policies** button, and search the **AmazonEC2ContainerRegistryFullAccess** policy. Mark the check box of the policy, and choose **Attach policy** to finish.

2. After you push the container, you can call the Amazon ECR image from anywhere in the SageMaker environment. Run the following code example in the next notebook cell.

If you want to use this training container with SageMaker Studio to use its visualization features, you can also run the following code in a Studio notebook cell to call the Amazon ECR image of your training container.

```
import boto3

account_id = boto3.client('sts').get_caller_identity().get('Account')
ecr_repository = 'sagemaker-byoc-test'
tag = ':latest'

region = boto3.session.Session().region_name

uri_suffix = 'amazonaws.com'
if region in ['cn-north-1', 'cn-northwest-1']:
    uri_suffix = 'amazonaws.com.cn'

byoc_image_uri = '{}.dkr.ecr.{}.{}'.format(account_id, region, uri_suffix,
                                           ecr_repository + tag)

byoc_image_uri
# This should return something like
# 111122223333.dkr.ecr.us-east-2.amazonaws.com/sagemaker-byoc-test:latest
```

3. Use the `ecr_image` retrieved from the previous step to configure a SageMaker estimator object. The following code sample configures a SageMaker estimator with the `byoc_image_uri` and initiates a training job on an Amazon EC2 instance.

### SageMaker Python SDK v1

```
import sagemaker
from sagemaker import get_execution_role
from sagemaker.estimator import Estimator
```

```

estimator = Estimator(image_name=byoc_image_uri,
                      role=get_execution_role(),
                      base_job_name='tf-custom-container-test-job',
                      instance_count=1,
                      instance_type='ml.p2.xlarge')

# start training
estimator.fit()

# deploy the trained model
predictor = estimator.deploy(1, instance_type)

```

### SageMaker Python SDK v2

```

import sagemaker
from sagemaker import get_execution_role
from sagemaker.estimator import Estimator

estimator = Estimator(image_uri=byoc_image_uri,
                      role=get_execution_role(),
                      base_job_name='tf-custom-container-test-job',
                      instance_count=1,
                      instance_type='ml.p2.xlarge')

# start training
estimator.fit()

# deploy the trained model
predictor = estimator.deploy(1, instance_type)

```

For a full example that shows how to test a custom container locally and push it to an Amazon ECR image, see the [Building Your Own TensorFlow Container](#) example notebook.

## Step 6: Clean up resources

### To clean up resources when done with the get started example

1. Open the [SageMaker console](#), choose the notebook instance **RunScriptNotebookInstance**, choose **Actions**, and choose **Stop**. It can take a few minutes for the instance to stop.
2. After the instance **Status** changes to **Stopped**, choose **Actions**, choose **Delete**, and then choose **Delete** in the dialog box. It can take a few minutes for the instance to be deleted. The notebook instance disappears from the table when it has been deleted.
3. Open the [Amazon S3 console](#) and delete the bucket that you created for storing model artifacts and the training dataset.
4. Open the [IAM console](#) and delete the IAM role. If you created permission policies, you can delete them, too.

#### Note

The Docker container shuts down automatically after it has run. You don't need to delete it.

## Adapting Your Own Inference Container

If none of the Amazon SageMaker prebuilt inference containers suffice for your situation, and you want to use your own Docker container, use the [SageMaker Inference Toolkit](#) to adapt your container to work with SageMaker hosting. To adapt your container to work with SageMaker hosting, create the inference code in one or more Python script files and a Dockerfile that imports the inference toolkit.

The inference code includes an inference handler, a handler service, and an endpoint. In this example, they are stored as three separate Python files. All three of these Python files must be in the same directory as your Dockerfile.

For an example Jupyter notebook that shows a complete example of extending a container by using the SageMaker inference toolkit, see [Amazon SageMaker Multi-Model Endpoints using your own algorithm container](#).

## Step 1: Create an Inference Handler

The SageMaker inference toolkit is built on the multi-model server (MMS). MMS expects a Python script that implements functions to load the model, pre-process input data, get predictions from the model, and process the output data in a model handler.

### The `model_fn` Function

The `model_fn` function is responsible for loading your model. It takes a `model_dir` argument that specifies where the model is stored. How you load your model depends on the framework you are using. There is no default implementation for the `model_fn` function. You must implement it yourself. The following simple example shows an implementation of a `model_fn` function that loads a PyTorch model:

```
def model_fn(self, model_dir):
    import torch

    logger.info('model_fn')
    device = "cuda" if torch.cuda.is_available() else "cpu"
    with open(os.path.join(model_dir, 'model.pth'), 'rb') as f:
        model = torch.jit.load(f)
    return model.to(device)
```

### The `input_fn` Function

The `input_fn` function is responsible for deserializing your input data so that it can be passed to your model. It takes input data and content type as parameters, and returns deserialized data. The SageMaker inference toolkit provides a default implementation that deserializes the following content types:

- JSON
- CSV
- Numpy array
- NPZ

If your model requires a different content type, or you want to preprocess your input data before sending it to the model, you must implement the `input_fn` function. The following example shows a simple implementation of the `input_fn` function.

```
from sagemaker_inference import content_types, decoder
def input_fn(self, input_data, content_type):
    """A default input_fn that can handle JSON, CSV and NPZ formats.

    Args:
        input_data: the request payload serialized in the content_type format
        content_type: the request content_type

    Returns: input_data deserialized into torch.FloatTensor or torch.cuda.FloatTensor
    depending if cuda is available.
    """
    return decoder.decode(input_data, content_type)
```

## The predict\_fn Function

The `predict_fn` function is responsible for getting predictions from the model. It takes the model and the data returned from `input_fn` as parameters, and returns the prediction. There is no default implementation for the `predict_fn`. You must implement it yourself. The following is a simple implementation of the `predict_fn` function for a PyTorch model.

```
def predict_fn(self, data, model):
    """A default predict_fn for PyTorch. Calls a model on data deserialized in
    input_fn.
    Runs prediction on GPU if cuda is available.

    Args:
        data: input data (torch.Tensor) for prediction deserialized by input_fn
        model: PyTorch model loaded in memory by model_fn

    Returns: a prediction
    """
    return model(input_data)
```

## The output\_fn Function

The `output_fn` function is responsible for serializing the data that the `predict_fn` function returns as a prediction. The SageMaker inference toolkit implements a default `output_fn` function that serializes Numpy arrays, JSON, and CSV. If your model outputs any other content type, or you want to perform other post-processing of your data before sending it to the user, you must implement your own `output_fn` function. The following shows a simple `output_fn` function for a PyTorch model.

```
from sagemaker_inference import encoder
def output_fn(self, prediction, accept):
    """A default output_fn for PyTorch. Serializes predictions from predict_fn to JSON,
    CSV or NPY format.

    Args:
        prediction: a prediction result from predict_fn
        accept: type which the output data needs to be serialized

    Returns: output data serialized
    """
    return encoder.encode(prediction, accept)
```

## Step 2: Implement a Handler Service

The handler service is executed by the model server. The handler service implements `initialize` and `handle` methods. The `initialize` method is invoked when the model server starts, and the `handle` method is invoked for all incoming inference requests to the model server. For more information, see [Custom Service](#) in the Multi-model server documentation. The following is an example of a handler service for a PyTorch model server.

```
from sagemaker_inference.default_handler_service import DefaultHandlerService
from sagemaker_inference.transformer import Transformer
from sagemaker_pytorch_serving_container.default_inference_handler import
    DefaultPytorchInferenceHandler

class HandlerService(DefaultHandlerService):
    """Handler service that is executed by the model server.
    Determines specific default inference handlers to use based on model being used.
    This class extends ``DefaultHandlerService``, which define the following:
```

```

        - The ``handle`` method is invoked for all incoming inference requests to the model
server.
        - The ``initialize`` method is invoked at model server start up.
Based on: https://github.com/awslabs/mxnet-model-server/blob/master/docs/
custom_service.md
"""
def __init__(self):
    transformer =
Transformer(default_inference_handler=DefaultPytorchInferenceHandler())
super(HandlerService, self).__init__(transformer=transformer)

```

## Step 3: Implement an Entrypoint

The entrypoint starts the model server by invoking the handler service. You specify the location of the entrypoint in your Dockerfile. The following is an example of an entrypoint.

```

from sagemaker_inference import model_server
model_server.start_model_server(handler_service=HANDLER_SERVICE)

```

## Step 4: Write a Dockerfile

In your Dockerfile, copy the model handler from step 2 and specify the Python file from the previous step as the entrypoint in your Dockerfile. The following is an example of the lines you can add to your Dockerfile to copy the model handler and specify the entrypoint. For a full example of a Dockerfile for an inference container, see [Dockerfile](#).

```

# Copy the default custom service file to handle incoming data and inference requests
COPY model_handler.py /home/model-server/model_handler.py

# Define an entrypoint script for the docker image
ENTRYPOINT ["python", "/usr/local/bin/entrypoint.py"]

```

## Step 5: Build and Register Your Container

Now you can build your container and register it in Amazon Elastic Container Registry (Amazon ECR). The following shell script from the sample notebook builds the container and uploads it to an Amazon ECR repository in your Amazon account.

### Note

SageMaker hosting supports using inference containers that are stored in repositories other than Amazon ECR. For information, see [Use a Private Docker Registry for Real-Time Inference Containers \(p. 2169\)](#).

The following shell script shows how to build and register a container.

```

%%sh

# The name of our algorithm
algorithm_name=demo-sagemaker-multimodel

cd container

account=$(aws sts get-caller-identity --query Account --output text)

# Get the region defined in the current configuration (default to us-west-2 if none
# defined)
region=$(aws configure get region)
region=${region:-us-west-2}

```

```
fullname="${account}.dkr.ecr.${region}.amazonaws.com/${algorithm_name}:latest"

# If the repository doesn't exist in ECR, create it.
aws ecr describe-repositories --repository-names "${algorithm_name}" > /dev/null 2>&1

if [ $? -ne 0 ]
then
    aws ecr create-repository --repository-name "${algorithm_name}" > /dev/null

# Get the login command from ECR and execute it directly
$(aws ecr get-login --region ${region} --no-include-email)

# Build the docker image locally with the image name and then push it to ECR
# with the full name.

docker build -q -t ${algorithm_name} .
docker tag ${algorithm_name} ${fullname}

docker push ${fullname}
```

You can now use this container to deploy endpoints in SageMaker. For an example of how to deploy an endpoint in SageMaker, see [Deploy the Model to SageMaker Hosting Services \(p. 63\)](#).

## Create a container with your own algorithms and models

If none of the existing SageMaker containers meet your needs and you don't have an existing container of your own, you may need to create a new Docker container. The following sections show how to create Docker containers with your training and inference algorithms for use with SageMaker.

### Topics

- [Use Your Own Training Algorithms \(p. 2160\)](#)
- [Use Your Own Inference Code \(p. 2166\)](#)

## Use Your Own Training Algorithms

This section explains how Amazon SageMaker interacts with a Docker container that runs your custom training algorithm. Use this information to write training code and create a Docker image for your training algorithms.

### Topics

- [How Amazon SageMaker Runs Your Training Image \(p. 2160\)](#)
- [How Amazon SageMaker Provides Training Information \(p. 2161\)](#)
- [Run Training with EFA \(p. 2164\)](#)
- [How Amazon SageMaker Signals Algorithm Success and Failure \(p. 2166\)](#)
- [How Amazon SageMaker Processes Training Output \(p. 2166\)](#)

## How Amazon SageMaker Runs Your Training Image

To configure a Docker container to run as an executable, use an `ENTRYPOINT` instruction in a Dockerfile. Note the following:

- For model training, Amazon SageMaker runs the container as follows:

```
docker run image train
```

SageMaker overrides any default `CMD` statement in a container by specifying the `train` argument after the image name. The `train` argument also overrides arguments that you provide using `CMD` in the Dockerfile.

- In your dockerfile, use the `exec` form of the `ENTRYPOINT` instruction:

```
ENTRYPOINT [ "executable", "param1", "param2", ... ]
```

For example:

```
ENTRYPOINT [ "python", "k-means-algorithm.py" ]
```

The `exec` form of the `ENTRYPOINT` instruction starts the executable directly, not as a child of `/bin/sh`. This enables it to receive signals like `SIGTERM` and `SIGKILL` from SageMaker APIs. Note the following:

- The [CreateTrainingJob](#) API has a stopping condition that directs SageMaker to stop model training after a specific time.
- The [StopTrainingJob](#) API issues the equivalent of the `docker stop`, with a 2-minute timeout command to gracefully stop the specified container:

```
docker stop -t120
```

The command attempts to stop the running container by sending a `SIGTERM` signal. After the 2-minute timeout, `SIGKILL` is sent and the containers are forcibly stopped. If the container handles the `SIGTERM` gracefully and exits within 120 seconds from receiving it, no `SIGKILL` is sent.

#### Note

If you want access to the intermediate model artifacts after SageMaker stops the training, add code to handle saving artifacts in your `SIGTERM` handler.

- If you plan to use GPU devices for model training, make sure that your containers are `nvidia-docker` compatible. Only the CUDA toolkit should be included on containers; don't bundle NVIDIA drivers with the image. For more information about `nvidia-docker`, see [NVIDIA/nvidia-docker](#).
- You can't use the `tini` initializer as your entry point in SageMaker containers because it gets confused by the `train` and `serve` arguments.
- `/opt/ml` and all sub-directories are reserved by SageMaker training. When building your algorithm's Docker image, please ensure you don't place any data required by your algorithm under them as the data may no longer be visible during training.

## How Amazon SageMaker Provides Training Information

This section explains how SageMaker makes training information, such as training data, hyperparameters, and other configuration information, available to your Docker container.

When you send a [CreateTrainingJob](#) request to SageMaker to start model training, you specify the Amazon Elastic Container Registry path of the Docker image that contains the training algorithm. You also specify the Amazon Simple Storage Service (Amazon S3) location where training data is stored and algorithm-specific parameters. SageMaker makes this information available to the Docker container so

that your training algorithm can use it. This section explains how we make this information available to your Docker container. For information about creating a training job, see [CreateTrainingJob](#). For more information on the way that SageMaker containers organize information, see [Using the SageMaker Training and Inference Toolkits \(p. 2149\)](#).

### Topics

- [Hyperparameters \(p. 2162\)](#)
- [Environment Variables \(p. 2162\)](#)
- [Input Data Configuration \(p. 2162\)](#)
- [Training Data \(p. 2163\)](#)
- [Distributed Training Configuration \(p. 2163\)](#)

## Hyperparameters

SageMaker makes the hyperparameters in a `CreateTrainingJob` request available in the Docker container in the `/opt/ml/input/config/hyperparameters.json` file.

## Environment Variables

The following environment variables are set in the container:

- `TRAINING_JOB_NAME` – Specified in the `TrainingJobName` parameter of the `CreateTrainingJob` request.
- `TRAINING_JOB_ARN` – The Amazon Resource Name (ARN) of the training job returned as the `TrainingJobArn` in the `CreateTrainingJob` response.
- All environment variables specified in the [Environment](#) parameter in the `CreateTrainingJob` request.

## Input Data Configuration

You specify data channel information in the `InputDataConfig` parameter in a `CreateTrainingJob` request. SageMaker makes this information available in the `/opt/ml/input/config/inputdataconfig.json` file in the Docker container.

For example, suppose that you specify three data channels (`train`, `evaluation`, and `validation`) in your request. SageMaker provides the following JSON:

```
{  
  "train" : {"ContentType": "trainingContentType",  
            "TrainingInputMode": "File",  
            "S3DistributionType": "FullyReplicated",  
            "RecordWrapperType": "None"},  
  "evaluation" : {"ContentType": "evalContentType",  
                 "TrainingInputMode": "File",  
                 "S3DistributionType": "FullyReplicated",  
                 "RecordWrapperType": "None"},  
  "validation" : {"TrainingInputMode": "File",  
                 "S3DistributionType": "FullyReplicated",  
                 "RecordWrapperType": "None"}  
}
```

### Note

SageMaker provides only relevant information about each data channel (for example, the channel name and the content type) to the container, as shown. `S3DistributionType` will be set as `FullyReplicated` if specify EFS or FSxLustre as input data sources.

## Training Data

The `TrainingInputMode` parameter in a `CreateTrainingJob` request specifies how to make data available for model training: in `FILE` mode or `PIPE` mode. Depending on the specified input mode, SageMaker does the following:

- **FILE mode**—SageMaker makes the data for the channel available in the `/opt/ml/input/data/channel_name` directory in the Docker container. For example, if you have three channels named `training`, `validation`, and `testing`, SageMaker makes three directories in the Docker container:
  - `/opt/ml/input/data/training`
  - `/opt/ml/input/data/validation`
  - `/opt/ml/input/data/testing`
- **Note**  
Channels that use file system data sources such as Amazon Elastic File System (Amazon EFS) and Amazon FSx must use FILE mode. Also to utilize an Amazon FSx file server, you must specify a path that begins with `/fsx`. If a file system is specified, the directory path provided in the channel is mounted at `/opt/ml/input/data/channel_name`.
- **PIPE mode**—SageMaker makes data for the channel available from the named pipe: `/opt/ml/input/data/channel_name_epoch_number`. For example, if you have three channels named `training`, `validation`, and `testing`, you will need to read from the following pipes:
  - `/opt/ml/input/data/training_0`, `/opt/ml/input/data/training_1`, ...
  - `/opt/ml/input/data/validation_0`, `/opt/ml/input/data/validation_1`, ...
  - `/opt/ml/input/data/testing_0`, `/opt/ml/input/data/testing_1`, ...

Read the pipes sequentially. For example, if you have a channel called `training`, read the pipes in this sequence:

1. Open `/opt/ml/input/data/training_0` in read mode and read it to end-of-file (EOF) or, if you are done with the first epoch, close the pipe file early.
2. After closing the first pipe file, look for `/opt/ml/input/data/training_1` and read it until you have completed the second epoch, and so on.

If the file for a given epoch doesn't exist yet, your code may need to retry until the pipe is created. There is no sequencing restriction across channel types. That is, you can read multiple epochs for the `training` channel, for example, and only start reading the `validation` channel when you are ready. Or, you can read them simultaneously if your algorithm requires that.

## Distributed Training Configuration

If you're performing distributed training with multiple containers, SageMaker makes information about all containers available in the `/opt/ml/input/config/resourceconfig.json` file.

To enable inter-container communication, this JSON file contains information for all containers. SageMaker makes this file available for both FILE and PIPE mode algorithms. The file provides the following information:

- `current_host`—The name of the current container on the container network. For example, `algo-1`. Host values can change at any time. Don't write code with specific values for this variable.
- `hosts`—The list of names of all containers on the container network, sorted lexicographically. For example, `["algo-1", "algo-2", "algo-3"]` for a three-node cluster. Containers can use these names to address other containers on the container network. Host values can change at any time. Don't write code with specific values for these variables.

- `network_interface_name`—The name of the network interface that is exposed to your container. For example, containers running the Message Passing Interface (MPI) can use this information to set the network interface name.
- Do not use the information in `/etc/hostname` or `/etc/hosts` because it might be inaccurate.
- Hostname information may not be immediately available to the algorithm container. We recommend adding a retry policy on hostname resolution operations as nodes become available in the cluster.

The following is an example file on node 1 in a three-node cluster:

```
{  
"current_host": "algo-1",  
"hosts": ["algo-1", "algo-2", "algo-3"],  
"network_interface_name": "eth1"  
}
```

## Run Training with EFA

SageMaker provides integration with [EFA](#) devices to accelerate High Performance Computing (HPC) and machine learning applications. This integration allows you to leverage an EFA device when running your distributed training jobs. You can add EFA integration to an existing Docker container that you bring to SageMaker. The following information outlines how to configure your own container to use an EFA device for your distributed training jobs.

### Prerequisites

Your container must satisfy the [SageMaker Training container specification](#).

### Install EFA and required packages

Your container must download and install the [EFA software](#). This allows your container to recognize the EFA device, and provides compatible versions of Libfabric and Open MPI.

Any tools like MPI and NCCL must be installed and managed inside the container to be used as part of your EFA-enabled training job. The following example shows how to modify the Dockerfile of your EFA-enabled container to install EFA, MPI, OFI, NCCL, and NCCL-TEST.

#### Note

When using PyTorch with EFA on your container, the NCCL version of your container should match the NCCL version of your PyTorch installation. To verify the PyTorch NCCL version, use the following command:

```
torch.cuda.nccl.version()
```

```
ARG OPEN_MPI_PATH=/opt/amazon/openmpi/  
ENV NCCL_VERSION=2.7.8  
ENV EFA_VERSION=1.11.2  
ENV BRANCH_OFI=1.1.1  
  
#####  
## EFA and MPI SETUP  
RUN cd $HOME \  
    && curl -O https://s3-us-west-2.amazonaws.com/aws-efa-installer/aws-efa-installer-  
${EFA_VERSION}.tar.gz \  
    && tar -xf aws-efa-installer-${EFA_VERSION}.tar.gz \  
    && cd aws-efa-installer \  
    && ./efa_installer.sh -y --skip-kmod -g \  
    && rm -rf aws-efa-installer
```

```

ENV PATH="$OPEN_MPI_PATH/bin:$PATH"
ENV LD_LIBRARY_PATH="$OPEN_MPI_PATH/lib/:$LD_LIBRARY_PATH"

#####
## NCCL, OFI, NCCL-TEST SETUP
RUN cd $HOME \
    && git clone https://github.com/NVIDIA/nccl.git -b v${NCCL_VERSION}-1 \
    && cd nccl \
    && make -j64 src.build BUILDDIR=/usr/local

RUN apt-get update && apt-get install -y autoconf
RUN cd $HOME \
    && git clone https://github.com/aws/aws-ofi-nccl.git -b v${BRANCH_OFI} \
    && cd aws-ofi-nccl \
    && ./autogen.sh \
    && ./configure --with-libfabric=/opt/amazon/efa \
        --with-mpi=/opt/amazon/openmpi \
        --with-cuda=/usr/local/cuda \
        --with-nccl=/usr/local --prefix=/usr/local \
    && make && make install

RUN cd $HOME \
    && git clone https://github.com/NVIDIA/nccl-tests \
    && cd nccl-tests \
    && make MPI=1 MPI_HOME=/opt/amazon/openmpi CUDA_HOME=/usr/local/cuda NCCL_HOME=/usr/local

```

## Considerations when creating your container

The EFA device is mounted to the container as `/dev/infiniband/uverbs0` under the list of device's accessible to the container. On P4d instances, the container has access to 4 EFA devices. The EFA devices can be found in the list of devices accessible to the container as:

- `/dev/infiniband/uverbs0`
- `/dev/infiniband/uverbs1`
- `/dev/infiniband/uverbs2`
- `/dev/infiniband/uverbs3`

To get information about hostname, peer hostnames, and network interface (for MPI) from the `resourceconfig.json` file provided to each container instances, see [Distributed Training Configuration](#). Your container handles regular TCP traffic among peers through the default Elastic Network Interfaces (ENI), while handling OFI (kernel bypass) traffic through the EFA device.

## Verify that your EFA device is recognized

To verify that the EFA device is recognized, run the following command from within your container.

```
/opt/amazon/efa/bin/fi_info -p efa
```

Your output should look similar to the following.

```

provider: efa
  fabric: EFA-fe80::e5:56ff:fe34:56a8
  domain: efa_0-rdm
  version: 2.0
  type: FI_EP_RDM
  protocol: FI_PROTO_EFA
provider: efa

```

```
fabric: EFA-fe80::e5:56ff:fe34:56a8
domain: efa_0-dgrm
version: 2.0
type: FI_EP_DGRAM
protocol: FI_PROTO_EFA
provider: efa;ofi_rxrd
fabric: EFA-fe80::e5:56ff:fe34:56a8
domain: efa_0-dgrm
version: 1.0
type: FI_EP_RDM
protocol: FI_PROTO_RXD
```

## Running a training job with EFA

Once you've created an EFA-enabled container, you can run a training job with EFA using a SageMaker Estimator the same way as you would with any other docker image. For more information on registering your container and using it for training, see [Adapting Your Own Training Container](#).

## How Amazon SageMaker Signals Algorithm Success and Failure

A training algorithm indicates whether it succeeded or failed using the exit code of its process.

A successful training execution should exit with an exit code of 0 and an unsuccessful training execution should exit with a non-zero exit code. These will be converted to `Completed` and `Failed` in the `TrainingJobStatus` returned by `DescribeTrainingJob`. This exit code convention is standard and is easily implemented in all languages. For example, in Python, you can use `sys.exit(1)` to signal a failure exit, and simply running to the end of the main routine will cause Python to exit with code 0.

In the case of failure, the algorithm can write a description of the failure to the failure file. See next section for details.

## How Amazon SageMaker Processes Training Output

As your algorithm runs in a container, it generates output including the status of the training job and model and output artifacts. Your algorithm should write this information to the following files, which are located in the container's `/output` directory. Amazon SageMaker processes the information contained in this directory as follows:

- `/opt/ml/output/failure`—If training fails, after all algorithm output (for example, logging) completes, your algorithm should write the failure description to this file. In a `DescribeTrainingJob` response, SageMaker returns the first 1024 characters from this file as `FailureReason`.
- `/opt/ml/model`—Your algorithm should write all final model artifacts to this directory. SageMaker copies this data as a single object in compressed tar format to the S3 location that you specified in the `CreateTrainingJob` request. If multiple containers in a single training job write to this directory they should ensure no file/directory names clash. SageMaker aggregates the result in a tar file and uploads to s3.

## Use Your Own Inference Code

You can use Amazon SageMaker to interact with Docker containers and run your own inference code in one of two ways:

- To use your own inference code with a persistent endpoint to get one prediction at a time, use SageMaker hosting services.

- To use your own inference code to get predictions for an entire dataset, use SageMaker batch transform.

#### Topics

- [Use Your Own Inference Code with Hosting Services \(p. 2167\)](#)
- [Use Your Own Inference Code with Batch Transform \(p. 2172\)](#)

## Use Your Own Inference Code with Hosting Services

This section explains how Amazon SageMaker interacts with a Docker container that runs your own inference code for hosting services. Use this information to write inference code and create a Docker image.

#### Topics

- [How SageMaker Runs Your Inference Image \(p. 2167\)](#)
- [How SageMaker Loads Your Model Artifacts \(p. 2168\)](#)
- [How Containers Serve Requests \(p. 2168\)](#)
- [How Your Container Should Respond to Inference Requests \(p. 2168\)](#)
- [How Your Container Should Respond to Health Check \(Ping\) Requests \(p. 2169\)](#)
- [Use a Private Docker Registry for Real-Time Inference Containers \(p. 2169\)](#)

## How SageMaker Runs Your Inference Image

To configure a container to run as an executable, use an `ENTRYPOINT` instruction in a Dockerfile. Note the following:

- For model inference, SageMaker runs the container as:

```
docker run image serve
```

SageMaker overrides default `CMD` statements in a container by specifying the `serve` argument after the `image` name. The `serve` argument overrides arguments that you provide with the `CMD` command in the Dockerfile.

- We recommend that you use the `exec` form of the `ENTRYPOINT` instruction:

```
ENTRYPOINT [ "executable", "param1", "param2" ]
```

For example:

```
ENTRYPOINT [ "python", "k_means_inference.py" ]
```

The `exec` form of the `ENTRYPOINT` instruction starts the executable directly, not as a child of `/bin/sh`. This enables it to receive signals like `SIGTERM` and `SIGKILL` from the SageMaker API operations, which is a requirement.

For example, when you use the [CreateEndpoint](#) API to create an endpoint, SageMaker provisions the number of ML compute instances required by the endpoint configuration, which you specify in the request. SageMaker runs the Docker container on those instances.

If you reduce the number of instances backing the endpoint (by calling the [UpdateEndpointWeightsAndCapacities](#) API), SageMaker runs a command to stop the Docker container on the instances that are being terminated. The command sends the SIGTERM signal, then it sends the SIGKILL signal thirty seconds later.

If you update the endpoint (by calling the [UpdateEndpoint](#) API), SageMaker launches another set of ML compute instances and runs the Docker containers that contain your inference code on them. Then it runs a command to stop the previous Docker containers. To stop a Docker container, command sends the SIGTERM signal, then it sends the SIGKILL signal 30 seconds later.

- SageMaker uses the container definition that you provided in your [CreateModel](#) request to set environment variables and the DNS hostname for the container as follows:
  - It sets environment variables using the `ContainerDefinition.Environment` string-to-string map.
  - It sets the DNS hostname using the `ContainerDefinition.ContainerHostname`.
- If you plan to use GPU devices for model inferences (by specifying GPU-based ML compute instances in your `CreateEndpointConfig` request), make sure that your containers are nvidia-docker compatible. Don't bundle NVIDIA drivers with the image. For more information about nvidia-docker, see [NVIDIA/nvidia-docker](#).
- You can't use the `tini` initializer as your entry point in SageMaker containers because it gets confused by the `train` and `serve` arguments.

## How SageMaker Loads Your Model Artifacts

In your [CreateModel](#) request, the container definition includes the `ModelDataUrl` parameter, which identifies the S3 location where model artifacts are stored. SageMaker uses this information to determine from where to copy the model artifacts. It copies the artifacts to the `/opt/ml/model` directory for use by your inference code.

The `ModelDataUrl` must point to a tar.gz file. Otherwise, SageMaker won't download the file.

If you trained your model in SageMaker, the model artifacts are saved as a single compressed tar file in Amazon S3. If you trained your model outside SageMaker, you need to create this single compressed tar file and save it in a S3 location. SageMaker decompresses this tar file into `/opt/ml/model` directory before your container starts.

## How Containers Serve Requests

Containers need to implement a web server that responds to `/invocations` and `/ping` on port 8080.

## How Your Container Should Respond to Inference Requests

To obtain inferences, the client application sends a POST request to the SageMaker endpoint. For more information, see the [InvokeEndpoint](#) API. SageMaker passes the request to the container, and returns the inference result from the container to the client. Note the following:

- SageMaker strips all `POST` headers except those supported by `InvokeEndpoint`. SageMaker might add additional headers. Inference containers must be able to safely ignore these additional headers.
- To receive inference requests, the container must have a web server listening on port 8080 and must accept `POST` requests to the `/invocations` endpoint.
- A customer's model containers must accept socket connection requests within 250 ms.

- A customer's model containers must respond to requests within 60 seconds. The model itself can have a maximum processing time of 60 seconds before responding to the /invocations. If your model is going to take 50-60 seconds of processing time, the SDK socket timeout should be set to be 70 seconds.

## How Your Container Should Respond to Health Check (Ping) Requests

The `CreateEndpoint` and `UpdateEndpoint` API calls result in SageMaker starting new inference containers. Soon after container startup, SageMaker starts sending periodic GET requests to the /ping endpoint.

The simplest requirement on the container is to respond with an HTTP 200 status code and an empty body. This indicates to SageMaker that the container is ready to accept inference requests at the /invocations endpoint.

If the container does not begin to pass health checks, by consistently responding with 200s, during the 4 minutes after startup, `CreateEndpoint` will fail, leaving the endpoint in a failed state, and the update requested by `UpdateEndpoint` will not be completed.

While the minimum bar is for the container to return a static 200, a container developer can use this functionality to perform deeper checks. The request timeout on /ping attempts is 2 seconds.

## Use a Private Docker Registry for Real-Time Inference Containers

Amazon SageMaker hosting enables you to use images stored in Amazon ECR to build your containers for real-time inference by default. Optionally, you can build containers for real-time inference from images in a private Docker registry. The private registry must be accessible from an Amazon VPC in your account. Models that you create based on the images stored in your private Docker registry must be configured to connect to the same VPC where the private Docker registry is accessible. For information about connecting your model to a VPC, see [Give SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC \(p. 2510\)](#).

Your Docker registry must be secured with a TLS certificate from a known public certificate authority (CA).

### Note

Your private Docker registry must allow inbound traffic from the security groups you specify in the VPC configuration for your model, so that SageMaker hosting is able to pull model images from your registry.

SageMaker can pull model images from DockerHub if there's a path to the open internet inside your VPC.

### Topics

- [Store Images in a Private Docker Registry other than Amazon Elastic Container Registry \(p. 2169\)](#)
- [Use an Image from a Private Docker Registry for Real-time Inference \(p. 2170\)](#)
- [Allow SageMaker to authenticate to a private Docker registry \(p. 2171\)](#)
- [Create the Lambda function \(p. 2171\)](#)
- [Give your execution role permission to Lambda \(p. 2172\)](#)
- [Create an interface VPC endpoint for Lambda \(p. 2172\)](#)

### [Store Images in a Private Docker Registry other than Amazon Elastic Container Registry](#)

To use a private Docker registry to store your images for SageMaker real-time inference, create a private registry that is accessible from your Amazon VPC. For information about creating a Docker registry,

see [Deploy a registry server](#) in the Docker documentation. The Docker registry must comply with the following:

- The registry must be a [Docker Registry HTTP API V2](#) registry.
- The Docker registry must be accessible from the same VPC that you specify in the `VpcConfig` parameter that you specify when you create your model.

## Use an Image from a Private Docker Registry for Real-time Inference

When you create a model and deploy it to SageMaker hosting, you can specify that it use an image from your private Docker registry to build the inference container. Specify this in the `ImageConfig` object in the `PrimaryContainer` parameter that you pass to a call to the [create\\_model](#) function.

### To use an image stored in your private Docker registry for your inference container

1. Create the image configuration object and specify a value of `Vpc` for the `RepositoryAccessMode` field.

```
image_config = {
    'RepositoryAccessMode': 'Vpc'
}
```

2. If your private Docker registry requires authentication, add a `RepositoryAuthConfig` object to the image configuration object. For the `RepositoryCredentialsProviderArn` field of the `RepositoryAuthConfig` object, specify the Amazon Resource Name (ARN) of an Amazon Lambda function that provides credentials that allows SageMaker to authenticate to your private Docker Registry. For information about how to create the Lambda function to provide authentication, see [Allow SageMaker to authenticate to a private Docker registry \(p. 2171\)](#).

```
image_config = {
    'RepositoryAccessMode': 'Vpc',
    'RepositoryAuthConfig': {
        'RepositoryCredentialsProviderArn':
            'arn:aws:lambda:Region:Acct:function:FunctionName'
    }
}
```

3. Create the primary container object that you want to pass to `create_model`, using the image configuration object that you created in the previous step.

```
primary_container = {
    'ContainerHostname': 'ModelContainer',
    'Image': 'myteam.myorg.com/docker-local/my-inference-image:latest',
    'ImageConfig': image_config
}
```

4. Specify the model name and the execution role that you want to pass to `create_model`.

```
model_name = 'vpc-model'
execution_role_arn = 'arn:aws:iam::123456789012:role/SageMakerExecutionRole'
```

5. Specify one or more security groups and subnets for the VPC configuration for your model. Your private Docker registry must allow inbound traffic from the security groups that you specify. The subnets that you specify must be in the same VPC as your private Docker registry.

```
vpc_config = {
    'SecurityGroupIds': ['sg-0123456789abcdef0'],
    'Subnets': ['subnet-0123456789abcdef0', 'subnet-0123456789abcdef1']
```

```
}
```

6. Get a Boto3 SageMaker client.

```
import boto3
sm = boto3.client('sagemaker')
```

7. Create the model by calling `create_model`, using the values you specified in the previous steps for the `PrimaryContainer` and `VpcConfig` parameters.

```
try:
    resp = sm.create_model(
        ModelName=model_name,
        PrimaryContainer=primary_container,
        ExecutionRoleArn=execution_role_arn,
        VpcConfig=vpc_config,
    )
except Exception as e:
    print(f'error calling CreateModel operation: {e}')
else:
    print(resp)
```

8. Finally, call `create_endpoint_config` and `create_endpoint` to create the hosting endpoint, using the model that you created in the previous step.

```
endpoint_config_name = 'my-endpoint-config'
sm.create_endpoint_config(
    EndpointConfigName=endpoint_config_name,
    ProductionVariants=[
        {
            'VariantName': 'MyVariant',
            'ModelName': model_name,
            'InitialInstanceCount': 1,
            'InstanceType': 'ml.t2.medium'
        },
    ],
)
endpoint_name = 'my-endpoint'
sm.create_endpoint(
    EndpointName=endpoint_name,
    EndpointConfigName=endpoint_config_name,
)
sm.describe_endpoint(EndpointName=endpoint_name)
```

## Allow SageMaker to authenticate to a private Docker registry

To pull an inference image from a private Docker registry that requires authentication, create an Amazon Lambda function that provides credentials, and provide the Amazon Resource Name (ARN) of the Lambda function when you call `create_model`. When SageMaker runs `create_model`, it calls the Lambda function that you specified to get credentials to authenticate to your Docker registry.

### Create the Lambda function

Create an Amazon Lambda function that returns a response with the following form:

```
def handler(event, context):
    response = {
        "Credentials": {"Username": "username", "Password": "password"}
```

```
    }
    return response
```

Depending on how you set up authentication for your private Docker registry, the credentials that your Lambda function returns can mean either of the following:

- If you set up your private Docker registry to use basic authentication, this is the *username* and *password* to authenticate to the registry.
- If you set up your private Docker registry to use bearer token authentication, the *username* and *password* are sent to your authorization server, which returns a Bearer token that can then be used to authenticate to the private Docker registry.

### Give your execution role permission to Lambda

The execution role that you use to call `create_model` must have permissions to call Amazon Lambda functions. Add the following to the permissions policy of your execution role.

```
{
  "Effect": "Allow",
  "Action": [
    "lambda:InvokeFunction"
  ],
  "Resource": [
    "arn:aws:lambda:*::*:function:*myLambdaFunction*"
  ]
}
```

Where `myLambdaFunction` is the name of your Lambda function. For information about editing a role permissions policy, see [Modifying a role permissions policy \(console\)](#) in the *Amazon Identity and Access Management User Guide*.

#### Note

An execution role with the `AmazonSageMakerFullAccess` managed policy attached to it has permission to call any Lambda function with **SageMaker** in its name.

### Create an interface VPC endpoint for Lambda

Create an interface endpoint so that your Amazon VPC can communicate with your Amazon Lambda function without sending traffic over the internet. For information about how to do this, see [Configuring interface VPC endpoints for Lambda](#) in the *Amazon Lambda Developer Guide*.

SageMaker hosting sends a request through your VPC to `lambda.region.amazonaws.com`, to call your Lambda function. If you choose Private DNS Name when you create your interface endpoint, Amazon Route 53 routes the call to the Lambda interface endpoint. If you use a different DNS provider, make sure to map `lambda.region.amazonaws.com` to your Lambda interface endpoint.

## Use Your Own Inference Code with Batch Transform

This section explains how Amazon SageMaker interacts with a Docker container that runs your own inference code for batch transform. Use this information to write inference code and create a Docker image.

### Topics

- [How SageMaker Runs Your Inference Image \(p. 2173\)](#)
- [How SageMaker Loads Your Model Artifacts \(p. 2173\)](#)
- [How Containers Serve Requests \(p. 2174\)](#)
- [How Your Container Should Respond to Inference Requests \(p. 2174\)](#)

- [How Your Container Should Respond to Health Check \(Ping\) Requests \(p. 2175\)](#)

## How SageMaker Runs Your Inference Image

To configure a container to run as an executable, use an `ENTRYPOINT` instruction in a Dockerfile. Note the following:

- For batch transforms, SageMaker runs the container as:

```
docker run image serve
```

SageMaker overrides default `CMD` statements in a container by specifying the `serve` argument after the image name. The `serve` argument overrides arguments that you provide with the `CMD` command in the Dockerfile.

- We recommend that you use the `exec` form of the `ENTRYPOINT` instruction:

```
ENTRYPOINT [ "executable", "param1", "param2" ]
```

For example:

```
ENTRYPOINT [ "python", "k_means_inference.py" ]
```

- SageMaker sets environment variables specified in [CreateModel](#) and [CreateTransformJob](#) on your container. Additionally, the following environment variables are populated:
  - `SAGEMAKER_BATCH` is always set to `true` when the container runs in Batch Transform.
  - `SAGEMAKER_MAX_PAYLOAD_IN_MB` is set to the largest size payload that is sent to the container via HTTP.
  - `SAGEMAKER_BATCH_STRATEGY` is set to `SINGLE_RECORD` when the container is sent a single record per call to invocations and `MULTI_RECORD` when the container gets as many records as will fit in the payload.
  - `SAGEMAKER_MAX_CONCURRENT_TRANSFORMS` is set to the maximum number of /invocations requests that can be opened simultaneously.

### Note

The last three environment variables come from the API call made by the user. If the user doesn't set values for them, they aren't passed. In that case, either the default values or the values requested by the algorithm (in response to the `/execution-parameters`) are used.

- If you plan to use GPU devices for model inferences (by specifying GPU-based ML compute instances in your [CreateTransformJob](#) request), make sure that your containers are nvidia-docker compatible. Don't bundle NVIDIA drivers with the image. For more information about nvidia-docker, see [NVIDIA/nvidia-docker](#).
- You can't use the `init` initializer as your entry point in SageMaker containers because it gets confused by the `train` and `serve` arguments.

## How SageMaker Loads Your Model Artifacts

In a [CreateModel](#) request, container definitions include the `ModelDataUrl` parameter, which identifies the location in Amazon S3 where model artifacts are stored. When you use SageMaker to run inferences, it uses this information to determine from where to copy the model artifacts. It copies the artifacts to the `/opt/ml/model` directory in the Docker container for use by your inference code.

The `ModelDataUrl` parameter must point to a tar.gz file. Otherwise, SageMaker can't download the file. If you train a model in SageMaker, it saves the artifacts as a single compressed tar file in Amazon S3. If you train a model in another framework, you need to store the model artifacts in Amazon S3 as a compressed tar file. SageMaker decompresses this tar file and saves it in the `/opt/ml/model` directory in the container before the batch transform job starts.

## How Containers Serve Requests

Containers must implement a web server that responds to invocations and ping requests on port 8080. For batch transforms, you have the option to set algorithms to implement execution-parameters requests to provide a dynamic runtime configuration to SageMaker. SageMaker uses the following endpoints:

- `ping`—Used to periodically check the health of the container. SageMaker waits for an HTTP 200 status code and an empty body for a successful ping request before sending an invocations request. You might use a ping request to load a model into memory to generate inference when invocations requests are sent.
- (Optional) `execution-parameters`—Allows the algorithm to provide the optimal tuning parameters for a job during runtime. Based on the memory and CPUs available for a container, the algorithm chooses the appropriate `MaxConcurrentTransforms`, `BatchStrategy`, and `MaxPayloadInMB` values for the job.

Before calling the invocations request, SageMaker attempts to invoke the execution-parameters request. When you create a batch transform job, you can provide values for the `MaxConcurrentTransforms`, `BatchStrategy`, and `MaxPayloadInMB` parameters. SageMaker determines the values for these parameters using this order of precedence:

1. The parameter values that you provide when you create the `CreateTransformJob` request.
2. The values that the model container returns when SageMaker invokes the execution-parameters endpoint>
3. The default parameter values, listed in the following table.

Parameter	Default Values
<code>MaxConcurrentTransforms</code>	1
<code>BatchStrategy</code>	<code>MULTI_RECORD</code>
<code>MaxPayloadInMB</code>	6

The response for a GET execution-parameters request is a JSON object with keys for `MaxConcurrentTransforms`, `BatchStrategy`, and `MaxPayloadInMB` parameters. This is an example of a valid response:

```
{
  "MaxConcurrentTransforms": 8,
  "BatchStrategy": "MULTI_RECORD",
  "MaxPayloadInMB": 6
}
```

## How Your Container Should Respond to Inference Requests

To obtain inferences, Amazon SageMaker sends a POST request to the inference container. The POST request body contains data from Amazon S3. Amazon SageMaker passes the request to the container, and returns the inference result from the container, saving the data from the response to Amazon S3.

To receive inference requests, the container must have a web server listening on port 8080 and must accept POST requests to the /invocations endpoint. Your model containers must respond to requests within 600 seconds.

## How Your Container Should Respond to Health Check (Ping) Requests

The simplest requirement on the container is to respond with an HTTP 200 status code and an empty body. This indicates to SageMaker that the container is ready to accept inference requests at the /invocations endpoint.

While the minimum bar is for the container to return a static 200, a container developer can use this functionality to perform deeper checks. The request timeout on /ping attempts is 2 seconds.

# Example Notebooks: Use Your Own Algorithm or Model

The following Jupyter notebooks show how to use your own algorithms or pretrained models from an Amazon SageMaker notebook instance. For links to the GitHub repositories with the prebuilt Dockerfiles for the TensorFlow, MXNet, Chainer, and PyTorch frameworks and instructions on using the Amazon SDK for Python (Boto3) estimators to run your own training algorithms on SageMaker Learner and your own models on SageMaker hosting, see [Prebuilt SageMaker Docker Images for Deep Learning \(p. 2135\)](#)

## Setup

1. Create a SageMaker notebook instance. For instructions on how to create and access Jupyter notebook instances, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#).
2. Open the notebook instance you created.
3. Choose the **SageMaker Examples** tab for a list of all SageMaker example notebooks.
4. Open the sample notebooks from the **Advanced Functionality** section in your notebook instance or from GitHub using the provided links. To open a notebook, choose its **Use** tab, then choose **Create copy**.

## Host Models Trained in Scikit-learn

To learn how to host models trained in Scikit-learn for making predictions in SageMaker by injecting them into first-party k-means and XGBoost containers, see the following sample notebooks.

- [kmeans\\_bring\\_your\\_own\\_model](#)
- [xgboost\\_bring\\_your\\_own\\_model](#)

## Package TensorFlow and Scikit-learn Models for Use in SageMaker

To learn how to package algorithms that you have developed in TensorFlow and scikit-learn frameworks for training and deployment in the SageMaker environment, see the following notebooks. They show you how to build, register, and deploy your own Docker containers using Dockerfiles.

- [tensorflow\\_bring\\_your\\_own](#)

- [scikit\\_bring\\_your\\_own](#)

## Train and Deploy a Neural Network on SageMaker

To learn how to train a neural network locally using MXNet or TensorFlow, and then create an endpoint from the trained model and deploy it on SageMaker, see the following notebooks. The MXNet model is trained to recognize handwritten numbers from the MNIST dataset. The TensorFlow model is trained to classify irises.

- [mxnet\\_mnist\\_byom](#)
- [tensorflow\\_iris\\_byom](#)

## Training Using Pipe Mode

To learn how to use a Dockerfile to build a container that calls the `train.py` script and uses pipe mode to custom train an algorithm, see the following notebook. In pipe mode, the input data is transferred to the algorithm while it is training. This can decrease training time compared to using file mode.

- [pipe\\_bring\\_your\\_own](#)

## Bring Your Own R Model

To learn how to use an R container to train and host a model with the R kernel installed in a notebook, see the following notebook. To take advantage of the Amazon SDK for Python (Boto3), we use Python within the notebook. You can achieve the same results in R by invoking command line arguments.

- [r\\_bring\\_your\\_own](#)

## Extend a Prebuilt PyTorch Container Image

To learn how to extend a prebuilt SageMaker PyTorch container image when you have additional functional requirements for your algorithm or model that the prebuilt Docker image doesn't support, see the following notebook.

- [pytorch\\_extending\\_our\\_containers](#)

## Train and Debug Training Jobs on a Custom Container

To learn how to train and debug training jobs using SageMaker Debugger, see the following notebook. A training script provided through this example uses the TensorFlow Keras ResNet 50 model and the CIFAR10 dataset. A Docker custom container is built with the training script and pushed to Amazon ECR. While the training job is running, Debugger collects tensor outputs and identifies debugging problems. With `smdebug` client library tools, you can set a `smdebug` trial object that calls the training job and debugging information, check the training and Debugger rule status, and retrieve tensors saved in an Amazon S3 bucket to analyze training issues.

- [build\\_your\\_own\\_container\\_with\\_debugger](#)

## Troubleshooting your Docker containers

The following are common errors that you might run into when using Docker containers with SageMaker. Each error is followed by a solution to the error.

- **Error: SageMaker has lost the Docker daemon.**

To fix this error, restart Docker using the following command.

```
sudo service docker restart
```

# SageMaker Workflows

You can manage your Amazon SageMaker training and inference workflows using Amazon SageMaker Studio and the Amazon SageMaker Python SDK. With the available tools, you can simplify your SageMaker process and integrate it into your existing project.

The following workflow technologies are supported.

- [Amazon SageMaker Model Building Pipelines \(p. 2178\)](#): SageMaker's tool for building and managing end-to-end ML pipelines.
- [Airflow Workflows](#): SageMaker APIs to export configurations for creating and managing Airflow workflows.
- [Kubernetes Orchestration \(p. 2263\)](#): SageMaker custom operators for your Kubernetes cluster, as well as custom components for Kubeflow Pipelines.
- [Amazon Step Functions](#): Create multi-step machine learning workflows in Python that orchestrate SageMaker infrastructure without having to provision your resources separately.

For more information on managing SageMaker training and inference, see [Amazon SageMaker Python SDK Workflows](#).

## Topics

- [Amazon SageMaker Model Building Pipelines \(p. 2178\)](#)
- [Automate MLOps with SageMaker Projects \(p. 2232\)](#)
- [Amazon SageMaker ML Lineage Tracking \(p. 2256\)](#)
- [Kubernetes Orchestration \(p. 2263\)](#)

## Amazon SageMaker Model Building Pipelines

Amazon SageMaker Model Building Pipelines is a tool for building machine learning pipelines that take advantage of direct SageMaker integration. Because of this integration, you can create a pipeline and set up SageMaker Projects for orchestration using a tool that handles much of the step creation and management for you. SageMaker Pipelines provides the following advantages over other Amazon workflow offerings:

### SageMaker Integration

SageMaker Pipelines is integrated directly with SageMaker, so you don't need to interact with any other Amazon services. You also don't need to manage any resources because SageMaker Pipelines is a fully managed service, which means that it creates and manages resources for you.

### SageMaker Python SDK Integration

Because SageMaker Pipelines is integrated with the SageMaker Python SDK, you can create your pipelines programmatically using a high-level Python interface that you might already be familiar with. To view the SageMaker Python SDK documentation, see [Pipelines](#).

### SageMaker Studio Integration

SageMaker Studio offers an environment to manage the end-to-end SageMaker Pipelines experience. Using Studio, you can bypass the Amazon console for your entire workflow management. For more information on managing SageMaker Pipelines from SageMaker Studio, see [View, Track, and Execute SageMaker Pipelines in SageMaker Studio \(p. 2217\)](#).

### Data Lineage Tracking

With SageMaker Pipelines you can track the history of your data within the pipeline execution. Amazon SageMaker ML Lineage Tracking lets you analyze where the data came from, where it was used as an input, and the outputs that were generated from it. For example, you can view the models created from an individual dataset, and you can view the datasets that went into creating an individual model. For more information, see [Amazon SageMaker ML Lineage Tracking \(p. 2256\)](#).

### Step Reuse

With SageMaker Pipelines, you can designate steps for caching. When a step is cached, it is indexed for reuse later if the same step is executed again. As a result, you can reuse the output from previous step executions of the same step in the same pipeline without having to run the step again. For more information on step caching, see [Caching Pipeline Steps \(p. 2192\)](#).

### Topics

- [SageMaker Pipelines Overview \(p. 2179\)](#)
- [Create and Manage SageMaker Pipelines \(p. 2201\)](#)

## SageMaker Pipelines Overview

An Amazon SageMaker Model Building Pipelines pipeline is a series of interconnected steps that is defined by a JSON pipeline definition. This pipeline definition encodes a pipeline using a directed acyclic graph (DAG). This DAG gives information on the requirements for and relationships between each step of your pipeline. The structure of a pipeline's DAG is determined by the data dependencies between steps. These data dependencies are created when the properties of a step's output are passed as the input to another step. The following image is an example of a pipeline DAG:



The following topics describe fundamental SageMaker Pipelines concepts. For a tutorial describing the implementation of these concepts, see [Create and Manage SageMaker Pipelines \(p. 2201\)](#).

#### Topics

- [Pipeline Structure \(p. 2181\)](#)
- [Access Management \(p. 2181\)](#)
- [Pipeline Parameters \(p. 2182\)](#)
- [Pipeline Steps \(p. 2183\)](#)
- [Property Files and JsonGet \(p. 2191\)](#)
- [Caching Pipeline Steps \(p. 2192\)](#)
- [Amazon EventBridge Integration \(p. 2193\)](#)

- [Amazon SageMaker Experiments Integration \(p. 2195\)](#)
- [SageMaker Pipelines Quotas \(p. 2198\)](#)
- [Troubleshooting Amazon SageMaker Model Building Pipelines \(p. 2200\)](#)

## Pipeline Structure

A pipeline instance is composed of a name, parameters, and steps. Pipeline names must be unique within an (account, region) pair. All parameters used in step definitions must be defined in the pipeline. Steps passed into the pipeline don't need to be listed in the order of execution because the steps themselves define the relationships between them using data dependencies. The SageMaker Pipelines service resolves the relationships between steps in the data dependency DAG to create a series of steps that the execution completes. The following is an example of a pipeline structure.

```
from sagemaker.workflow.pipeline import Pipeline

pipeline_name = f"AbalonePipeline"
pipeline = Pipeline(
    name=pipeline_name,
    parameters=[
        processing_instance_type,
        processing_instance_count,
        training_instance_type,
        model_approval_status,
        input_data,
        batch_data,
    ],
    steps=[step_process, step_train, step_eval, step_cond],
)
```

## Access Management

The following sections describe the Amazon Identity and Access Management (IAM) requirements for Amazon SageMaker Model Building Pipelines. For an example of how you can implement these permissions, see [Prerequisites \(p. 2203\)](#).

### Topics

- [Pipeline Role Permissions \(p. 2181\)](#)
- [Pipeline Step Permissions \(p. 2182\)](#)
- [Service Control Policies with Pipelines \(p. 2182\)](#)

## Pipeline Role Permissions

Your pipeline requires an IAM pipeline execution role that is passed to SageMaker Pipelines when you create a pipeline. The role for the SageMaker instance that is creating the pipeline must have the `iam:PassRole` permission for the pipeline execution role in order to pass it. For more information on IAM roles, see [IAM Roles](#).

Your pipeline execution role requires the following permissions:

- To pass any role to a SageMaker job within a pipeline, the `iam:PassRole` permission for the role that is being passed.
- `Create and Describe` permissions for each of the job types in the pipeline.
- Amazon S3 permissions to use the `JsonGet` function. You control access to your Amazon S3 resources using resource-based policies and identity-based policies. A resource-based policy is applied to your

Amazon S3 bucket and grants SageMaker Pipelines access to the bucket. An identity-based policy gives your pipeline the ability to make Amazon S3 calls from your account. For more information on resource-based policies and identity-based policies, see [Identity-based policies and resource-based policies](#).

```
{  
    "Action": [  
        "s3:GetObject",  
        "s3:HeadObject"  
    ],  
    "Resource": "arn:aws:s3:::<your-bucket-arn>/*",  
    "Effect": "Allow"  
}
```

## Pipeline Step Permissions

SageMaker Pipelines include steps that run SageMaker jobs. In order for the pipeline steps to run these jobs, they require an IAM role in your account that provides access for the needed resource. This role is passed to the SageMaker service principal by your pipeline. For more information on IAM roles, see [IAM Roles](#).

By default, each step takes on the pipeline execution role. You can optionally pass a different role to any of the steps in your pipeline. This ensures that the code in each step does not have the ability to impact resources used in other steps unless there is a direct relationship between the two steps specified in the pipeline definition. You pass these roles when defining the processor or estimator for your step. For examples of how to include these roles in these definitions, see the [SageMaker Python SDK documentation](#).

## Service Control Policies with Pipelines

Service control policies (SCPs) are a type of organization policy that you can use to manage permissions in your organization. SCPs offer central control over the maximum available permissions for all accounts in your organization. By using SageMaker Pipelines within your organization, you can ensure that data scientists manage your pipeline executions without having to interact with the Amazon console.

If you're using a VPC with your SCP that restricts access to Amazon S3, you need to take steps to allow your pipeline to access other Amazon S3 resources.

To allow SageMaker Pipelines to access Amazon S3 outside of your VPC with the `JsonGet` function, update your organization's SCP to ensure that the role using SageMaker Pipelines can access Amazon S3. To do this, create an exception for roles that are being used by the SageMaker Pipelines executor via the pipeline execution role using a principal tag and condition key.

### To allow SageMaker Pipelines to access Amazon S3 outside of your VPC

1. Create a unique tag for your pipeline execution role following the steps in [Tagging IAM users and roles](#).
2. Grant an exception in your SCP using the `Aws:PrincipalTag` IAM condition key for the tag you created. For more information, see [Creating, updating, and deleting service control policies](#).

## Pipeline Parameters

You can introduce variables into your pipeline definition using parameters. Parameters that you define can be referenced throughout your pipeline definition. Parameters have a default value, which you can override by specifying parameter values when starting a pipeline execution. The default value must be an instance matching the parameter type. All parameters used in step definitions must be defined in

your pipeline definition. Amazon SageMaker Model Building Pipelines supports the following parameter types:

- `ParameterString` – Representing a `str` Python type.
- `ParameterInteger` – Representing an `int` Python type.
- `ParameterFloat` – Representing a `float` Python type.

Parameters take the following format:

```
<parameter> = <parameter_type>(  
    name="<parameter_name>",  
    default_value=<default_value>  
)
```

The following shows an example parameter implementation:

```
from sagemaker.workflow.parameters import (  
    ParameterInteger,  
    ParameterString,  
    ParameterFloat  
)  
  
processing_instance_count = ParameterInteger(  
    name="ProcessingInstanceCount",  
    default_value=1  
)
```

You pass the parameter when creating your pipeline as follows:

```
pipeline = Pipeline(  
    name=pipeline_name,  
    parameters=[  
        processing_instance_count  
    steps=[step_process]  
)
```

You can also pass a parameter value that differs from the default value to a pipeline execution as follows:

```
execution = pipeline.start(  
    parameters=dict(  
        ProcessingInstanceType="ml.c5.xlarge",  
        ModelApprovalStatus="Approved"  
    )  
)
```

## Pipeline Steps

SageMaker Pipelines are composed of steps. These steps define the actions that the pipeline takes, and the relationships between steps using properties.

### Topics

- [Step Types \(p. 2184\)](#)
- [Step Properties \(p. 2189\)](#)

- [Data Dependency Between Steps \(p. 2189\)](#)
- [Custom Dependency Between Steps \(p. 2190\)](#)
- [Use a Custom Image in a Step \(p. 2191\)](#)

## Step Types

The following describes the requirements of each step type and provides an example implementation of the step. These are not functional implementations because they don't provide the resource and inputs needed. For a tutorial that implements these steps, see [Create and Manage SageMaker Pipelines \(p. 2201\)](#).

Amazon SageMaker Model Building Pipelines support the following step types:

- [Processing \(p. 2184\)](#)
- [Training \(p. 2185\)](#)
- [Transform \(p. 2186\)](#)
- [CreateModel \(p. 2186\)](#)
- [RegisterModel \(p. 2186\)](#)
- [Condition \(p. 2187\)](#)
- [Callback \(p. 2187\)](#)

### Processing Step

You use a processing step to create a processing job for data processing. For more information on processing jobs, see [Process Data and Evaluate Models](#).

A processing step requires a processor, a Python script that defines the processing code, outputs for processing, and job arguments. The following example shows how to create a `ProcessingStep` definition. For more information on processing step requirements, see the [sagemaker.workflow.steps.ProcessingStep](#) documentation.

```
from sagemaker.sklearn.processing import SKLearnProcessor

sklearn_processor = SKLearnProcessor(framework_version='0.20.0',
                                      role=<role>,
                                      instance_type='ml.m5.xlarge',
                                      instance_count=1)
```

```
from sagemaker.processing import ProcessingInput, ProcessingOutput
from sagemaker.workflow.steps import ProcessingStep

step_process = ProcessingStep(
    name="AbaloneProcess",
    processor=sklearn_processor,
    inputs=[
        ProcessingInput(source=<input_data>, destination="/opt/ml/processing/input"),
    ],
    outputs=[
        ProcessingOutput(output_name="train", source="/opt/ml/processing/train"),
        ProcessingOutput(output_name="validation", source="/opt/ml/processing/validation"),
        ProcessingOutput(output_name="test", source="/opt/ml/processing/test")
    ],
    code="abalone/preprocessing.py"
)
```

### Pass runtime parameters

You can pass runtime parameters to a processing step using the [get\\_run\\_args](#) method of the [Amazon SageMaker Python SDK](#). This allows you to use processors besides SKLearnProcessor such as PySparkProcessor and SparkJarProcessor.

The following example shows how to pass runtime parameters from a PySpark processor to a ProcessingStep.

```
from sagemaker.spark.processing import PySparkProcessor

pyspark_processor = PySparkProcessor(framework_version='2.4',
                                     role=<role>,
                                     instance_type='ml.m5.xlarge',
                                     instance_count=1)

from sagemaker.processing import ProcessingInput, ProcessingOutput

run_args = pyspark_processor.get_run_args(
    "preprocess.py",
    inputs=[
        ProcessingInput(source=<input_data>, destination="/opt/ml/processing/input"),
    ],
    outputs=[
        ProcessingOutput(output_name="train", source="/opt/ml/processing/train"),
        ProcessingOutput(output_name="validation", source="/opt/ml/processing/validation"),
        ProcessingOutput(output_name="test", source="/opt/ml/processing/test")
    ],
    arguments=None
)

from sagemaker.workflow.steps import ProcessingStep

step_process = ProcessingStep(
    name="AbaloneProcess",
    processor=pyspark_processor,
    inputs=run_args.inputs,
    outputs=run_args.outputs,
    job_arguments=run_args.arguments,
    code=run_args.code
)
```

### Training Step

You use a training step to create a training job to train a model. For more information on training jobs, see [Train a Model with Amazon SageMaker](#).

A training step requires an estimator, and training and validation data inputs. The following example shows how to create a TrainingStep definition. For more information on Training step requirements, see the [sagemaker.workflow.steps.TrainingStep](#) documentation.

```
from sagemaker.inputs import TrainingInput
from sagemaker.workflow.steps import TrainingStep

step_train = TrainingStep(
    name="TrainAbaloneModel",
    estimator=xgb_train,
    inputs={
        "train": TrainingInput(
```

```
s3_data=step_process.properties.ProcessingOutputConfig.Outputs[
    "train"
].S3Output.S3Uri,
content_type="text/csv"
),
"validation": TrainingInput(
    s3_data=step_process.properties.ProcessingOutputConfig.Outputs[
        "validation"
].S3Output.S3Uri,
content_type="text/csv"
)
}
)
```

## Transform Step

You use a transform step for batch transformation to run inference on an entire dataset. For more information on batch transformation, see [Run Batch Transforms with Inference Pipelines](#).

A transform step requires a transformer, and the data to run batch transformation on. The following example shows how to create a `TransformStep` definition. For more information on `TransformStep` requirements, see the [sagemaker.workflow.steps.TransformStep](#) documentation.

```
from sagemaker.inputs import TransformInput
from sagemaker.workflow.steps import TransformStep

step_transform = TransformStep(
    name="AbaloneTransform",
    transformer=transformer,
    inputs=TransformInput(data=batch_data)
)
```

## CreateModel Step

You use a `CreateModel` step to create a SageMaker Model. For more information on SageMaker Models, see [Train a Model with Amazon SageMaker](#).

A `CreateModel` step requires model artifacts, and information on the SageMaker instance type that you need to use to create the model. The following example shows how to create a `CreateModel` step definition. For more information on `CreateModel` step requirements, see the [sagemaker.workflow.steps.CreateModelStep](#) documentation.

```
from sagemaker.workflow.steps import CreateModelStep

step_create_model = CreateModelStep(
    name="AbaloneCreateModel",
    model=model,
    inputs=inputs
)
```

## RegisterModel Step

You use a `RegisterModel` step to register a model to a model group. For more information on registering models, see [Register and Deploy Models with Model Registry \(p. 2003\)](#).

A `RegisterModel` step requires an estimator, model data output from training, and a model package group name to associate the model package with. The following example shows how to create a `RegisterModel` definition. For more information on `RegisterModel` step requirements, see the [sagemaker.workflow.step\\_collections.RegisterModel](#) documentation.

```
from sagemaker.workflow.step_collections import RegisterModel

step_register = RegisterModel(
    name="AbaloneRegisterModel",
    estimator=xgb_train,
    model_data=step_train.properties.ModelArtifacts.S3ModelArtifacts,
    content_types=["text/csv"],
    response_types=["text/csv"],
    inference_instances=["ml.t2.medium", "ml.m5.xlarge"],
    transform_instances=["ml.m5.xlarge"],
    model_package_group_name=model_package_group_name,
    approval_status=model_approval_status,
    model_metrics=model_metrics
)
```

## Condition Step

You use a condition step to evaluate the condition of step properties to assess which action should be taken next in the pipeline.

A condition step requires a list of conditions, and a list of steps to execute if the condition evaluates to `true` and a list of steps to execute if the condition evaluates to `false`. The following example shows how to create a Condition step definition. For more information on Condition step requirements, see the [sagemaker.workflow.condition\\_step.ConditionStep](#) documentation.

## Limitations

- SageMaker Pipelines doesn't support the use of nested condition steps. You can't pass a condition step as the input for another condition step.
- A condition step can't use identical steps in both branches. If you need the same step functionality in both branches, duplicate the step and give it a different name.

```
from sagemaker.workflow.conditions import ConditionLessThanOrEqualTo
from sagemaker.workflow.condition_step import (
    ConditionStep,
    JsonGet
)

cond_lte = ConditionLessThanOrEqualTo(
    left=JsonGet(
        step=step_eval,
        property_file=evaluation_report,
        json_path="regression_metrics.mse.value"
    ),
    right=6.0
)

step_cond = ConditionStep(
    name="AbaloneMSECond",
    conditions=[cond_lte],
    if_steps=[step_register, step_create_model, step_transform],
    else_steps=[]
)
```

## Callback Step

You can use a callback step to incorporate additional processes and Amazon services into your workflow that aren't directly provided by Amazon SageMaker Model Building Pipelines. When a callback step runs, the following procedure occurs:

- SageMaker Pipelines sends a message to a customer-specified Amazon Simple Queue Service (Amazon SQS) queue. The message contains a SageMaker Pipelines-generated token and a customer-supplied list of input parameters. After sending the message, SageMaker Pipelines waits for a response from the customer.
- The customer retrieves the message from the Amazon SQS queue and starts their custom process.
- When the process finishes, the customer calls one of the following APIs and submits the SageMaker Pipelines-generated token:
  - [SendPipelineExecutionStepSuccess](#) – along with a list of output parameters
  - [SendPipelineExecutionStepFailure](#) – along with a failure reason
- The API call causes SageMaker Pipelines to either continue the pipeline execution or fail the execution.

For more information on Callback step requirements, see the [sagemaker.workflow.callback\\_step](#) documentation.

### Stopping Behavior

A pipeline execution won't stop while a callback step is running.

When you call [StopPipelineExecution](#) on a pipeline execution with a running callback step, SageMaker Pipelines sends an additional Amazon SQS message to the specified SQS queue. The body of the SQS message contains a "Status" field which is set to "Stopping".

You should add logic to your Amazon SQS message consumer to take any needed action (for example, resource cleanup) upon receipt of the message followed by a call to [SendPipelineExecutionStepSuccess](#) or [SendPipelineExecutionStepFailure](#).

Only when SageMaker Pipelines receives one of these calls will it stop the pipeline execution.

#### Important

Callback steps were introduced in Amazon SageMaker Python SDK v2.45.0 and Amazon SageMaker Studio v3.6.2. You must update Studio before you use a callback step or the pipeline DAG doesn't display. To update Studio, see [Update SageMaker Studio \(p. 117\)](#).

The following sample demonstrates an implementation of the preceding procedure.

```
from sagemaker.workflow.callback_step import CallbackStep

step_callback = CallbackStep(
    name="MyCallbackStep",
    sqs_queue_url="https://sqs.us-east-2.amazonaws.com/012345678901/MyCallbackQueue",
    inputs={...},
    outputs=[...]
)

callback_handler_code = '
    import boto3
    import json

    def handler(event, context):
        sagemaker_client=boto3.client("sagemaker")

        for record in event["Records"]:
            payload=json.loads(record["body"])
            token=payload["token"]

            # Custom processing

            # Call SageMaker to complete the step
            sagemaker_client.send_pipeline_execution_step_success(
```

```
        CallbackToken=token,  
        OutputParameters={...}  
    )  
,
```

## Stopping Behavior

A pipeline execution won't stop while a callback step is running.

When you call [StopPipelineExecution](#) on a pipeline execution with a running callback step, SageMaker Pipelines sends an additional Amazon SQS message to the specified SQS queue. The body of the SQS message contains a "Status" field which is set to "Stopping".

You should add logic to your Amazon SQS message consumer to take any needed action (for example, resource cleanup) upon receipt of the message followed by a call to [SendPipelineExecutionStepSuccess](#) or [SendPipelineExecutionStepFailure](#).

Only when SageMaker Pipelines receives one of these calls will it stop the pipeline execution.

## Step Properties

The `properties` attribute is used to add data dependencies between steps in the pipeline. These data dependencies are then used by SageMaker Pipelines to construct the DAG from the pipeline definition. These properties can be referenced as placeholder values and are resolved at runtime.

The `properties` attribute of a SageMaker Pipelines step matches the object returned by a `Describe` call for the corresponding SageMaker job type. For each job type, the `Describe` call returns the following response object:

- `ProcessingStep` – [DescribeProcessingJob](#)
- `TrainingStep` – [DescribeTrainingJob](#)
- `TransformStep` – [DescribeTransformJob](#)

## Data Dependency Between Steps

You define the structure of your DAG by specifying the data relationships between steps. To create data dependencies between steps, pass the properties of one step as the input to another step in the pipeline. The step receiving the input isn't started until after the step providing the input finishes execution.

A data dependency uses JsonPath notation in the following format. This format traverses the JSON property file, which means you can append as many `<property>` instances as needed to reach the desired nested property in the file. For more information on JsonPath notation, see the [JsonPath repo](#).

```
<step_name>.properties.<property>.<property>
```

The following shows how to specify an Amazon S3 bucket using the `ProcessingOutputConfig` property of a processing step.

```
step_process.properties.ProcessingOutputConfig.Outputs["train_data"].S3Output.S3Uri
```

To create the data dependency, pass the bucket to a training step as follows.

```
step_train = TrainingStep(  
    name="CensusTrain",
```

```
estimator=sklearn_train,
inputs=TrainingInput(
    s3_data=step_process.properties.ProcessingOutputConfig.Outputs[
        "train_data"].S3Output.S3Uri
)
)
```

## Custom Dependency Between Steps

When you specify a data dependency, SageMaker Pipelines provides the data connection between the steps. Alternatively, one step can access the data from a previous step without directly using SageMaker Pipelines. In this case, you can create a custom dependency that tells SageMaker Pipelines not to start a step until after another step has finished executing. You create a custom dependency by specifying a step's `DependsOn` attribute.

As an example, the following defines a step C that starts only after both step A and step B finish executing.

```
{
    'Steps': [
        {'Name': 'A', ...},
        {'Name': 'B', ...},
        {'Name': 'C', 'DependsOn': ['A', 'B']}
    ]
}
```

SageMaker Pipelines throws a validation exception if the dependency would create a cyclic dependency.

The following example creates a training step that starts after a processing step finishes executing.

```
processing_step = ProcessingStep(...)
training_step = TrainingStep(...)

training_step.add_depends_on([processing_step])
```

The following example creates a training step that doesn't start until two different processing steps finish executing.

```
processing_step_1 = ProcessingStep(...)
processing_step_2 = ProcessingStep(...)

training_step = TrainingStep(...)

training_step.add_depends_on([processing_step_1, processing_step_2])
```

The following provides an alternate way to create the custom dependency.

```
training_step.add_depends_on([processing_step_1])
training_step.add_depends_on([processing_step_2])
```

The following example creates a training step that receives input from one processing step and waits for a different processing step to finish executing.

```
processing_step_1 = ProcessingStep(...)
processing_step_2 = ProcessingStep(...)
```

```
training_step = TrainingStep(  
    ...  
    inputs=TrainingInput(  
        s3_data=processing_step_1.properties.ProcessingOutputConfig.Outputs[  
            "train_data"  
        ].S3Output.S3Uri  
    )  
  
    training_step.add_depends_on([processing_step_2])
```

The following example shows how to retrieve a string list of the custom dependencies of a step.

```
custom_dependencies = training_step.depends_on()
```

## Use a Custom Image in a Step

You can use any of the available SageMaker [Deep Learning Container images](#) when you create a step in your pipeline.

You can also create a step using SageMaker S3 applications. A SageMaker S3 application is a tar.gz bundle with one or more Python scripts that can run within that bundle. For more information on application package bundling, see [Deploying directly from model artifacts](#).

You can also use your own container with pipeline steps. Because you can't create an image from within SageMaker Studio, you must create your image using another method before using it with Amazon SageMaker Model Building Pipelines.

To use your own container when creating the steps for your pipeline, include the image URI in the estimator definition. For more information on using your own container with SageMaker, see [Using Docker Containers with SageMaker](#).

## Property Files and JsonGet

You use property files to store information from the output of a processing step. This is particularly useful when analyzing the results of a processing step to decide how a conditional step should be executed. The `JsonGet` function processes a property file and enables you to use JsonPath notation to query the property JSON file. For more information on JsonPath notation, see the [JsonPath repo](#).

To store a property file for later use, you must first create a `PropertyFile` instance with the following format. The `path` parameter is the name of the JSON file that the property file is saved to. `output_name` must match the `output_name` of the `ProcessingOutput` that you define in your processing step. This enables the property file to capture the `ProcessingOutput` in the step.

```
from sagemaker.workflow.properties import PropertyFile  
  
<property_file_instance> = PropertyFile(  
    name=<property_file_name>,  
    output_name=<processingoutput_output_name>,  
    path=<path_to_json_file>  
)
```

When you create your `ProcessingStep` instance, add the `property_files` parameter to list all of the parameter files that the Amazon SageMaker Model Building Pipelines service must index. This saves the property file for later use.

```
property_files=[<property_file_instance>]
```

To use your property file in a condition step, add the `property_file` to the condition that you pass to your condition step as follows. This enables you to query the JSON file for your desired property using the `json_path` parameter.

```
cond_lte = ConditionLessThanOrEqualTo(
    left=JsonGet(
        step=step_eval,
        property_file=<property_file_instance>,
        json_path="mse"
    ),
    right=6.0
)
```

## Caching Pipeline Steps

When you use step signature caching, before SageMaker Pipelines executes a step, it attempts to find a previous execution of a step that was called with the same arguments. SageMaker Pipelines checks that the call signatures are identical. Pipelines doesn't check whether the data or code that the arguments point to has changed. If a previous execution is found, a cache hit is created. Pipelines then propagates the values from the cache hit during execution, rather than recomputing the step.

Step caching only considers successful executions, so no failed executions are ever reused. When multiple successful executions exist within the timeout period, Pipelines uses the result for the most recent successful execution. If no successful executions match in the timeout period, Pipelines won't reuse any steps. If the executor finds a cache hit for a previous step execution that is still in progress, both steps continue executing and update the cache, if they're successful.

You must opt-in to step caching, otherwise it is off by default. When you enable step caching, you must also define a timeout. This timeout defines how old a previous execution can be to be considered for reuse.

Step caching is only scoped for individual pipelines, so you can't reuse a step from another pipeline. Even if there is a step signature match in the other pipeline, the step is not reused.

Step caching is available for the following step types:

- Training
- Processing
- Transform

## Enabling Step Caching

To enable step caching, you must add a `CacheConfig` property to the step definition.

`CacheConfig` properties use the following format in the pipeline definition file.

```
{
  "CacheConfig": {
    "Enabled": false,
    "ExpireAfter": "<time>"
  }
}
```

The `Enabled` field may be true or false. `ExpireAfter` is a string that defines the timeout period. Any ISO 8601 duration string is a valid `ExpireAfter` value. The `ExpireAfter` duration can contain a

year, month, week, day, hour, and minute value. Each value consists of a number followed by a letter indicating the duration unit it is for. For example:

- "30d" = Thirty days
- "5y" = Five years
- "T16m" = 16 minutes
- "30dT5h" = 30 days and five hours.

The following example shows how to enable caching for a training step using the Amazon SageMaker Python SDK.

```
cache_config = CacheConfig(enable_caching=True, expire_after="PT1H")

step_train = TrainingStep(
    name="TrainAbaloneModel",
    estimator=xgb_train,
    inputs=inputs,
    cache_config=cache_config
)
```

## Amazon EventBridge Integration

You can schedule your Amazon SageMaker Model Building Pipelines executions using [Amazon EventBridge](#). Amazon SageMaker Model Building Pipelines is supported as a target in [Amazon EventBridge](#). This allows you to trigger the execution of your model building pipeline based on any event in your event bus. EventBridge enables you to automate your pipeline executions and respond automatically to events such as training job or endpoint status changes. Events include a new file being uploaded to your Amazon S3 bucket, a change in status of your Amazon SageMaker endpoint due to drift, and [Amazon Simple Notification Service \(SNS\) topics](#).

The following SageMaker Pipelines actions can be automatically triggered:

- `StartPipelineExecution`

For more information on scheduling SageMaker jobs, see [Automating SageMaker with Amazon EventBridge](#).

### Schedule a Pipeline with Amazon EventBridge

To start a pipeline execution with Amazon CloudWatch Events, you must create an EventBridge rule. When you create a rule for events, you specify a target action to take when EventBridge receives an event that matches the rule. When an event matches the rule, EventBridge sends the event to the specified target and triggers the action defined in the rule.

The following tutorials show how to schedule a pipeline execution with EventBridge using the EventBridge console or the Amazon CLI.

#### Prerequisites

- A role that can be assumed by EventBridge with the `SageMaker::StartPipelineExecution` permission. This role can be created automatically if you create a rule from the EventBridge console, otherwise you need to create this role yourself. For information on creating a SageMaker role, see [SageMaker Roles](#).
- An Amazon SageMaker Pipeline to schedule. To create an Amazon SageMaker Pipeline, see [Define a Pipeline](#).

## Create an EventBridge rule using the EventBridge console

The following procedure shows how to create an EventBridge rule using the EventBridge console.

1. Navigate to the [EventBridge console](#).
2. Select Rules on the left hand side.
3. Select Create Rule.
4. Enter a name and description for your rule.
5. Select how you want this rule to be triggered. You have the following choices for your rule:
  - **Event pattern:** Your rule is triggered when an event matching the pattern occurs. You can choose a pre-defined pattern that matches a certain type of event, or you can create a custom pattern. If you select a pre-defined pattern, you can edit the pattern to customize it. For more information on Event patterns, see [Event Patterns in CloudWatch Events](#).
  - **Schedule:** Your rule is triggered regularly on a specified schedule. You can use a fixed-rate schedule that triggers regularly for a specified number of minutes, hour, or weeks. You can also use a [cron expression](#) to create a more fine-grained schedule, such as "the first Monday of each month at 8am." Schedule is not supported on a custom or partner event bus.
6. Select your desired Event bus.
7. Select the target(s) to invoke when an event matches your event pattern or when the schedule is triggered. You can add up to 5 targets per rule. Select SageMaker Pipeline in the target drop-down.
8. Select the pipeline you want to trigger from the pipeline drop-down.
9. Add parameters to pass to your pipeline execution using a name and value pair. Parameter values can be static or dynamic. For more information on Amazon SageMaker Pipeline parameters, see [AWS::Events::Rule SagemakerPipelineParameters](#).
  - Static values are passed to the pipeline execution every time the pipeline is triggered. For example, if `{ "Name": "Instance_type", "Value": "ml.4xlarge" }` is specified in the parameter list, then it will be passed as a parameter in `StartPipelineExecutionRequest` every time EventBridge triggers the pipeline.
  - Dynamic values are specified using a json path. EventBridge parses the value from an event payload, then passes it to the pipeline execution. For example: `$.detail.param.value`
10. Select the role to use for this rule. You can either use an existing role or create a new one.
11. (Optional) Add tags.
12. Select Create to finalize your rule.

Your rule is now in effect and ready to trigger your pipeline executions.

## Create an EventBridge rule using the Amazon CLI

The following procedure shows how to create an EventBridge rule using the Amazon CLI.

1. Create a rule to be triggered. When creating an EventBridge rule using the Amazon CLI, you have two options for how your rule is triggered, event pattern and schedule.
  - **Event pattern:** Your rule is triggered when an event matching the pattern occurs. You can choose a pre-defined pattern that matches a certain type of event, or you can create a custom pattern. If you select a pre-defined pattern, you can edit the pattern to customize it. You can create a rule with event pattern using the following command:

```
aws events put-rule --name <RULE_NAME> ----event-pattern <YOUR_EVENT_PATTERN> --description <RULE_DESCRIPTION> --role-arn <ROLE_TO_EXECUTE_PIPELINE> --tags <TAGS>
```

- **Schedule:** Your rule is triggered regularly on a specified schedule. You can use a fixed-rate schedule that triggers regularly for a specified number of minutes, hour, or weeks. You can also use a cron expression to create a more fine-grained schedule, such as “the first Monday of each month at 8am.” Schedule is not supported on a custom or partner event bus. You can create a rule with schedule using the following command:

```
aws events put-rule --name <RULE_NAME> --schedule-expression <YOUR_CRON_EXPRESSION>
--description <RULE_DESCRIPTION> --role-arn <ROLE_TO_EXECUTE_PIPELINE> --tags <TAGS>
```

2. Add target(s) to invoke when an event matches your event pattern or when the schedule is triggered. You can add up to 5 targets per rule. For each target, you must specify:
  - ARN: The resource ARN of your pipeline.
  - Role ARN: The ARN of the role EventBridge should assume to execute the pipeline.
  - Parameters: Amazon SageMaker pipeline parameters to pass.
3. Run the following command to pass a Amazon SageMaker pipeline as a target to your rule using [put-targets](#) :

```
aws events put-targets --rule <RULE_NAME> --event-bus-name <EVENT_BUS_NAME>
--targets "[{\\"Id\\": <ID>, \\"Arn\\": <RESOURCE_ARN>, \\"RoleArn\\": <ROLE_ARN>,
\\\"SageMakerPipelineParameter\\": { \\"SageMakerParameterList\\": [{{\\\"Name\\": <NAME>,
\\\"Value\\": <VALUE>}}]} }]"
```

## Amazon SageMaker Experiments Integration

Amazon SageMaker Model Building Pipelines is closely integrated with Amazon SageMaker Experiments. By default, when SageMaker Pipelines creates and executes a pipeline, the following SageMaker Experiments entities are created if they don't exist:

- An experiment for the pipeline
- A trial for every execution of the pipeline
- A trial component that's added to the trial for each SageMaker job created in a pipeline execution step

You can compare metrics such as model training accuracy across multiple pipeline executions just as you can compare such metrics across multiple trials of a SageMaker model training experiment.

The following sample shows the relevant parameters of the [Pipeline](#) class in the [Amazon SageMaker Python SDK](#).

```
Pipeline(
    name="MyPipeline",
    parameters=[...],
    pipeline_experiment_config=PipelineExperimentConfig(
        ExecutionVariables.PIPELINE_NAME,
        ExecutionVariables.PIPELINE_EXECUTION_ID
    ),
    steps=[...]
)
```

If you don't want an experiment and trial created for the pipeline, set `pipeline_experiment_config` to `None`.

### Note

Experiments integration was introduced in the Amazon SageMaker Python SDK v2.41.0.

The following naming rules apply based on what you specify for the `ExperimentName` and `TrialName` parameters of `pipeline_experiment_config`:

- If you don't specify `ExperimentName`, the pipeline name is used for the experiment name.

If you do specify `ExperimentName`, it's used for the experiment name. If an experiment with that name exists, the pipeline-created trials are added to the existing experiment. If an experiment with that name doesn't exist, a new experiment is created.

- If you don't specify `TrialName`, the pipeline execution ID is used for the trial name.

If you do specify `TrialName`, it's used for the trial name. If a trial with that name exists, the pipeline-created trial components are added to the existing trial. If a trial with that name doesn't exist, a new trial is created.

#### Note

The experiment entities aren't deleted when the pipeline that created the entities is deleted. You can use the SageMaker Experiments API to delete the entities. For more information, see [Clean Up Amazon SageMaker Experiment Resources \(p. 1572\)](#).

For information about how to view the SageMaker Experiment entities associated with a pipeline, see [View Experiment Entities Created by SageMaker Pipelines \(p. 2224\)](#). For more information on SageMaker Experiments, see [Manage Machine Learning with Amazon SageMaker Experiments \(p. 1553\)](#).

The following sections show examples of the previous rules and how they are represented in the pipeline definition file. For more information on pipeline definition files, see [SageMaker Pipelines Overview \(p. 2179\)](#).

#### Topics

- [Default Behavior \(p. 2196\)](#)
- [Disable Experiments Integration \(p. 2197\)](#)
- [Specify a Custom Experiment Name \(p. 2197\)](#)
- [Specify a Custom Trial Name \(p. 2198\)](#)

## Default Behavior

### Create a pipeline

The `pipeline_experiment_config` is omitted. `ExperimentName` defaults to the pipeline name. `TrialName` defaults to the execution ID.

```
pipeline_name = f"MyPipeline"
pipeline = Pipeline(
    name=pipeline_name,
    parameters=[...],
    steps=[step_train]
)
```

### Pipeline definition file

```
{
  "Version": "2020-12-01",
  "Parameters": [
    {
      "Name": "InputDataSource"
    },

```

```
{
    "Name": "InstanceCount",
    "Type": "Integer",
    "DefaultValue": 1
}
],
"PipelineExperimentConfig": {
    "ExperimentName": {"Get": "Execution.PipelineName"},
    "TrialName": {"Get": "Execution.PipelineExecutionId"}
},
"Steps": [...]
}
```

## Disable Experiments Integration

### Create a pipeline

The `pipeline_experiment_config` is set to `None`.

```
pipeline_name = f"MyPipeline"
pipeline = Pipeline(
    name=pipeline_name,
    parameters=[...],
    pipeline_experiment_config=None,
    steps=[step_train]
)
```

### Pipeline definition file

This is the same as the preceding default example, without the `PipelineExperimentConfig`.

## Specify a Custom Experiment Name

A custom experiment name is used. The trial name is set to the execution ID, as with the default behavior.

### Create a pipeline

```
pipeline_name = f"MyPipeline"
pipeline = Pipeline(
    name=pipeline_name,
    parameters=[...],
    pipeline_experiment_config=PipelineExperimentConfig(
        "CustomExperimentName",
        ExecutionVariables.PIPELINE_EXECUTION_ID
    ),
    steps=[step_train]
)
```

### Pipeline definition file

```
{
    ...,
    "PipelineExperimentConfig": {
        "ExperimentName": "CustomExperimentName",
        "TrialName": {"Get": "Execution.PipelineExecutionId"}
    },
    "Steps": [...]
}
```

## Specify a Custom Trial Name

A custom trial name is used and appended with the execution ID. The experiment name is set to the pipeline name, as with the default behavior.

### Create a pipeline

```
pipeline_name = f"MyPipeline"
pipeline = Pipeline(
    name=pipeline_name,
    parameters=[...],
    pipeline_experiment_config=PipelineExperimentConfig(
        ExecutionVariables.PIPELINE_NAME,
        Join(on="-", values=["CustomTrialName", ExecutionVariables.PIPELINE_EXECUTION_ID])
    ),
    steps=[step_train]
)
```

### Pipeline definition file

```
{
  ...
  "PipelineExperimentConfig": {
    "ExperimentName": {"Get": "Execution.PipelineName"},
    "TrialName": {
      "On": "-",
      "Values": [
        "CustomTrialName",
        {"Get": "Execution.PipelineExecutionId"}
      ]
    }
  },
  "Steps": [...]
}
```

## SageMaker Pipelines Quotas

The following tables list the quotas for SageMaker Pipelines resources and requirements for pipeline creation.

### Pipeline Quotas

<b>Pipeline Property</b>	<b>Quota</b>
Maximum number of pipelines	5000

### Executions Quotas

<b>Execution Property</b>	<b>Quota</b>
Maximum execution time	28 days
Maximum number of concurrent pipeline executions per account	200
Maximum number of concurrent pipeline executions per pipeline	200

## Step Quotas

<b>Step Property</b>	<b>Quota</b>
Maximum number of steps per pipeline	50
Maximum step name length	64 characters
Step name regular expression pattern	([A-Za-z0-9\\-_-])*

## Parameters Quotas

<b>Parameters Property</b>	<b>Quota</b>
Maximum number of parameters per pipeline	200
Maximum parameter name length	64 characters
Maximum parameter name regular expression pattern	([A-Za-z0-9\\-_-])*
Maximum parameter description length	4096 characters
Maximum number of parameter enum values	16 distinct values
Maximum parameter enum value length	2048 characters
Maximum string default value limit	2048 characters

## Condition Step Quotas

<b>Condition Step Property</b>	<b>Quota</b>
Maximum number of Conditions per ConditionStep	200
Maximum number of Steps in If-List	20
Maximum number of Steps in Else-List	20
Maximum number of Conditions in an Or-list	200

## Property Files Quotas

<b>Property Files Property</b>	<b>Quota</b>
Maximum number of PropertyFiles in a pipeline	10
Maximum number of JsonGet functions in a pipeline	200
Maximum size of the property file	2MB

## Metadata Quotas

Metadata Property	Quota
Maximum metadata size	20 key-value pairs
Maximum metadata key size	128 characters
Metadata key regular expression pattern	([A-Za-z0-9\-\_])*
Maximum metadata value size	1024 characters
Metadata value regular expression pattern	[\\p{M}\\p{L}\\p{S}\\p{S}\\p{N}\\p{P}\\s]

## Troubleshooting Amazon SageMaker Model Building Pipelines

When using Amazon SageMaker Model Building Pipelines, you might run into issues for various reasons. This topic provides information about common errors and how to resolve them.

### Pipeline Definition Issues

Your pipeline definition might not be formatted correctly. This can result in your execution failing or your job being inaccurate. These errors can be caught when the pipeline is created or when an execution occurs. If your definition doesn't validate, SageMaker Pipelines returns an error message identifying the character where the JSON file is malformed. To fix this problem, review the steps created using the SageMaker Python SDK for accuracy.

You can only include steps in a pipeline definition once. Because of this, steps cannot exist as part of a Condition step *and* a pipeline in the same pipeline.

### Examining Pipeline Logs

You can view the status of your steps using the following command:

```
execution.list_steps()
```

Each step includes the following information:

- The ARN of the entity launched by the pipeline – Such as SageMaker job ARN, model ARN, or model package ARN.
- The failure reason – Includes a brief explanation of the step failure.
- Condition step evaluation – If the step is a condition step, it includes whether the condition is evaluated to true or false.
- The CacheHit – If the execution is reusing a previous job execution, it lists the source execution.

You can also view the error messages and logs in the SageMaker Studio interface. For information about how to see the logs in Studio, see [View a Pipeline Execution \(p. 2221\)](#).

### Missing Permissions

Correct permissions are required for the role that creates the pipeline execution, and the steps that create each of the jobs in your pipeline execution. Without these permissions, you may not be able to submit your pipeline execution or run your SageMaker jobs as expected. To ensure that your permissions are properly set up, see [Access Management \(p. 2181\)](#).

### Job Execution Errors

You may run into issues when executing your steps because of issues in the scripts that define the functionality of your SageMaker jobs. Each job has a set of CloudWatch logs. To view these logs from SageMaker Studio, see [View a Pipeline Execution \(p. 2221\)](#). For information about using CloudWatch logs with SageMaker, see [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 2534\)](#).

### Property File Errors

You may have issues when incorrectly implementing property files with your pipeline. To ensure that your implementation of property files works as expected, see [Property Files and `JsonGet` \(p. 2191\)](#).

## Create and Manage SageMaker Pipelines

You can use Amazon SageMaker Model Building Pipelines to create end-to-end workflows that manage and deploy SageMaker jobs. SageMaker Pipelines comes with SageMaker Python SDK integration, so you can build each step of your pipeline using a Python-based interface.

After your pipeline is deployed, you can view the directed acyclic graph (DAG) for your pipeline and manage your executions using Amazon SageMaker Studio. Using SageMaker Studio, you can get information about your current and historical pipelines, compare executions, see the DAG for your executions, get metadata information, and more. To learn how to view pipelines from SageMaker Studio, see [View, Track, and Execute SageMaker Pipelines in SageMaker Studio \(p. 2217\)](#).

### Topics

- [Define a Pipeline \(p. 2201\)](#)
- [Run a pipeline \(p. 2214\)](#)
- [View, Track, and Execute SageMaker Pipelines in SageMaker Studio \(p. 2217\)](#)

## Define a Pipeline

To orchestrate your workflows with Amazon SageMaker Model Building Pipelines, you need to generate a directed acyclic graph (DAG) in the form of a JSON pipeline definition. The following image is a representation of the pipeline DAG that you create in this tutorial:



You can generate your JSON pipeline definition using the SageMaker Python SDK. The following tutorial shows how to generate a pipeline definition for a pipeline that solves a regression problem to determine the age of an abalone based on its physical measurements. For a Jupyter notebook that includes the content in this tutorial that you can run, see [Orchestrating Jobs with Amazon SageMaker Model Building Pipelines](#).

#### Topics

- [Prerequisites \(p. 2203\)](#)
- [Create a Pipeline \(p. 2203\)](#)

## Prerequisites

To run the following tutorial you must do the following:

- Set up your notebook instance as outlined in [Create a notebook instance](#). This gives your role permissions to read and write to Amazon S3, and create training, batch transform, and processing jobs in SageMaker.
- Grant your notebook permissions to get and pass its own role as shown in [Modifying a role permissions policy](#). Add the following JSON snippet to attach this policy to your role. Replace <your-role-arn> with the ARN used to create your notebook instance.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "iam:GetRole",  
                "iam:PassRole"  
            ],  
            "Resource": "<your-role-arn>"  
        }  
    ]  
}
```

- Trust the SageMaker service principal by following the steps in [Modifying a role trust policy](#). Add the following statement fragment to the trust relationship of your role:

```
{  
    "Sid": "",  
    "Effect": "Allow",  
    "Principal": {  
        "Service": "sagemaker.amazonaws.com"  
    },  
    "Action": "sts:AssumeRole"  
}
```

## Set Up Your Environment

Create a new SageMaker session using the following code block. This returns the role ARN for the session. This role ARN should be the execution role ARN that you set up as a prerequisite.

```
import boto3  
import sagemaker  
import sagemaker.session  
  
region = boto3.Session().region_name  
sagemaker_session = sagemaker.session.Session()  
role = sagemaker.get_execution_role()  
default_bucket = sagemaker_session.default_bucket()  
model_package_group_name = f"AbaloneModelPackageGroupName"
```

## Create a Pipeline

Run the following steps from your SageMaker notebook instance to create a pipeline including steps for preprocessing, training, evaluation, conditional evaluation, and model registration.

## Step 1: Download the Dataset

This notebook uses the UCI Machine Learning Abalone Dataset. The dataset contains the following features:

- `length` – The longest shell measurement of the abalone.
- `diameter` – The diameter of the abalone perpendicular to its length.
- `height` – The height of the abalone with meat in the shell.
- `whole_weight` – The weight of the whole abalone.
- `shucked_weight` – The weight of the meat removed from the abalone.
- `viscera_weight` – The weight of the abalone viscera after bleeding.
- `shell_weight` – The weight of the abalone shell after meat removal and drying.
- `sex` – The sex of the abalone. One of 'M', 'F', or 'I', where 'I' is an infant abalone.
- `rings` – The number of rings in the abalone shell.

The number of rings in the abalone shell is a good approximation for its age using the formula `age=rings + 1.5`. However, obtaining this number is a time-consuming task. You must cut the shell through the cone, stain the section, and count the number of rings through a microscope. However, the other physical measurements are easier to determine. This notebook uses the dataset to build a predictive model of the variable `rings` using the other physical measurements.

### To download the dataset

1. Download the dataset into your account's default Amazon S3 bucket.

```
!mkdir -p data
local_path = "data/abalone-dataset.csv"

s3 = boto3.resource("s3")
s3.Bucket(f"sagemaker-servicecatalog-seedcode-{region}").download_file(
    "dataset/abalone-dataset.csv",
    local_path
)

base_uri = f"s3://{default_bucket}/abalone"
input_data_uri = sagemaker.s3.S3Uploader.upload(
    local_path=local_path,
    desired_s3_uri=base_uri,
)
print(input_data_uri)
```

2. Download a second dataset for batch transformation after your model is created.

```
local_path = "data/abalone-dataset-batch"

s3 = boto3.resource("s3")
s3.Bucket(f"sagemaker-servicecatalog-seedcode-{region}").download_file(
    "dataset/abalone-dataset-batch",
    local_path
)

base_uri = f"s3://{default_bucket}/abalone"
batch_data_uri = sagemaker.s3.S3Uploader.upload(
    local_path=local_path,
    desired_s3_uri=base_uri,
)
print(batch_data_uri)
```

## Step 2: Define Pipeline Parameters

This code block defines the following parameters for your pipeline:

- `processing_instance_type` – The `ml.*` instance type of the processing jobs.
- `processing_instance_count` – The instance count of the processing job.
- `training_instance_type` – The `ml.*` instance type of the training jobs.
- `input_data` – The Amazon S3 location of the input data.
- `batch_data` – The Amazon S3 location of the input data for batch transformation.
- `model_approval_status` – The approval status to register the trained model with for CI/CD. For more information, see [Automate MLOps with SageMaker Projects \(p. 2232\)](#).

```
from sagemaker.workflow.parameters import (
    ParameterInteger,
    ParameterString,
)

processing_instance_count = ParameterInteger(
    name="ProcessingInstanceCount",
    default_value=1
)
processing_instance_type = ParameterString(
    name="ProcessingInstanceType",
    default_value="ml.m5.xlarge"
)
training_instance_type = ParameterString(
    name="TrainingInstanceType",
    default_value="ml.m5.xlarge"
)
model_approval_status = ParameterString(
    name="ModelApprovalStatus",
    default_value="PendingManualApproval"
)
input_data = ParameterString(
    name="InputData",
    default_value=input_data_uri,
)
batch_data = ParameterString(
    name="BatchData",
    default_value=batch_data_uri,
)
```

## Step 3: Define a Processing Step for Feature Engineering

This section shows how to create a processing step to prepare the data from the dataset for training.

### To create a processing step

1. Create a directory for the processing script.

```
!mkdir -p abalone
```

2. Create a file in the `/abalone` directory named `preprocessing.py` with the following content. This preprocessing script is passed in to the processing step for execution on the input data. The training step then uses the preprocessed training features and labels to train a model, and the evaluation step uses the trained model and preprocessed test features and labels to evaluate the model. The script uses `scikit-learn` to do the following:

- Fill in missing `sex` categorical data and encode it so it's suitable for training.
- Scale and normalize all numerical fields except for `rings` and `sex`.
- Split the data into training, test, and validation datasets.

```
%>%%writefile abalone/preprocessing.py
import argparse
import os
import requests
import tempfile
import numpy as np
import pandas as pd

from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder

# Because this is a headerless CSV file, specify the column names here.
feature_columns_names = [
    "sex",
    "length",
    "diameter",
    "height",
    "whole_weight",
    "shucked_weight",
    "viscera_weight",
    "shell_weight",
]
label_column = "rings"

feature_columns_dtype = {
    "sex": str,
    "length": np.float64,
    "diameter": np.float64,
    "height": np.float64,
    "whole_weight": np.float64,
    "shucked_weight": np.float64,
    "viscera_weight": np.float64,
    "shell_weight": np.float64
}
label_column_dtype = {"rings": np.float64}

def merge_two_dicts(x, y):
    z = x.copy()
    z.update(y)
    return z

if __name__ == "__main__":
    base_dir = "/opt/ml/processing"

    df = pd.read_csv(
        f"{base_dir}/input/abalone-dataset.csv",
        header=None,
        names=feature_columns_names + [label_column],
        dtype=merge_two_dicts(feature_columns_dtype, label_column_dtype)
    )
    numeric_features = list(feature_columns_names)
    numeric_features.remove("sex")
```

```

        numeric_transformer = Pipeline(
            steps=[
                ("imputer", SimpleImputer(strategy="median")),
                ("scaler", StandardScaler())
            ]
        )

        categorical_features = ["sex"]
        categorical_transformer = Pipeline(
            steps=[
                ("imputer", SimpleImputer(strategy="constant", fill_value="missing")),
                ("onehot", OneHotEncoder(handle_unknown="ignore"))
            ]
        )

        preprocess = ColumnTransformer(
            transformers=[
                ("num", numeric_transformer, numeric_features),
                ("cat", categorical_transformer, categorical_features)
            ]
        )

        y = df.pop("rings")
        X_pre = preprocess.fit_transform(df)
        y_pre = y.to_numpy().reshape(len(y), 1)

        X = np.concatenate((y_pre, X_pre), axis=1)

        np.random.shuffle(X)
        train, validation, test = np.split(X, [int(.7*len(X)), int(.85*len(X))])

        pd.DataFrame(train).to_csv(f"{base_dir}/train/train.csv", header=False,
        index=False)
        pd.DataFrame(validation).to_csv(f"{base_dir}/validation/validation.csv",
        header=False, index=False)
        pd.DataFrame(test).to_csv(f"{base_dir}/test/test.csv", header=False, index=False)
    
```

3. Create an instance of an `SKLearnProcessor` to pass in to the processing step.

```

from sagemaker.sklearn.processing import SKLearnProcessor

framework_version = "0.23-1"

sklearn_processor = SKLearnProcessor(
    framework_version=framework_version,
    instance_type=processing_instance_type,
    instance_count=processing_instance_count,
    base_job_name="sklearn-abalone-process",
    role=role,
)
    
```

4. Create a processing step. This step takes in the `SKLearnProcessor`, the input and output channels, and the `preprocessing.py` script that you created. This is very similar to a processor instance's `run` method in the SageMaker Python SDK. The `input_data` parameter passed into `ProcessingStep` is the input data of the step itself. This input data is used by the processor instance when it runs.

Note the "train," "validation," and "test" named channels specified in the output configuration for the processing job. Step Properties such as these can be used in subsequent steps and resolve to their runtime values at execution.

```
from sagemaker.processing import ProcessingInput, ProcessingOutput
from sagemaker.workflow.steps import ProcessingStep

step_process = ProcessingStep(
    name="AbaloneProcess",
    processor=sklearn_processor,
    inputs=[
        ProcessingInput(source=input_data, destination="/opt/ml/processing/input"),
    ],
    outputs=[
        ProcessingOutput(output_name="train", source="/opt/ml/processing/train"),
        ProcessingOutput(output_name="validation", source="/opt/ml/processing/validation"),
        ProcessingOutput(output_name="test", source="/opt/ml/processing/test")
    ],
    code="abalone/preprocessing.py",
)
```

#### Step 4: Define a Training step

This section shows how to use the SageMaker [XGBoost Algorithm](#) to train a logistic regression model on the training data output from the processing steps.

#### To define a training step

1. Specify the model path where you want to save the models from training.

```
model_path = f"s3://{default_bucket}/AbaloneTrain"
```

2. Configure an estimator for the XGBoost algorithm and the input dataset. The `training_instance_type` is passed into the estimator. A typical training script loads data from the input channels, configures training with hyperparameters, trains a model, and saves a model to `model_dir` so that it can be hosted later. SageMaker uploads the model to Amazon S3 in the form of a `model.tar.gz` at the end of the training job.

```
from sagemaker.estimator import Estimator

image_uri = sagemaker.image_uris.retrieve(
    framework="xgboost",
    region=region,
    version="1.0-1",
    py_version="py3",
    instance_type=training_instance_type,
)
xgb_train = Estimator(
    image_uri=image_uri,
    instance_type=training_instance_type,
    instance_count=1,
    output_path=model_path,
    role=role,
)
xgb_train.set_hyperparameters(
    objective="reg:linear",
    num_round=50,
    max_depth=5,
    eta=0.2,
    gamma=4,
    min_child_weight=6,
```

```

        subsample=0.7,
        silent=0
    )

```

3. Create a `TrainingStep` using the estimator instance and properties of the `ProcessingStep`. In particular, pass in the `S3Uri` of the "train" and "validation" output channel to the `TrainingStep`.

```

from sagemaker.inputs import TrainingInput
from sagemaker.workflow.steps import TrainingStep

step_train = TrainingStep(
    name="AbaloneTrain",
    estimator=xgb_train,
    inputs={
        "train": TrainingInput(
            s3_data=step_process.properties.ProcessingOutputConfig.Outputs[
                "train"
            ].S3Output.S3Uri,
            content_type="text/csv"
        ),
        "validation": TrainingInput(
            s3_data=step_process.properties.ProcessingOutputConfig.Outputs[
                "validation"
            ].S3Output.S3Uri,
            content_type="text/csv"
        )
    }
)

```

## Step 5: Define a Processing Step for Model Evaluation

This section shows how to create a processing step to evaluate the accuracy of the model. The result of this model evaluation is used in the condition step to determine which execute path to take.

### To define a processing step for model evaluation

1. Create a file in the `/abalone` directory named `evaluation.py`. This script is used in a processing step to perform model evaluation. It takes a trained model and the test dataset as input, then produces a JSON file containing classification evaluation metrics. These metrics include a precision, recall, and F1 score for each label, and accuracy and ROC curve for the model.

```

%%writefile abalone/evaluation.py
import json
import pathlib
import pickle
import tarfile
import joblib
import numpy as np
import pandas as pd
import xgboost

from sklearn.metrics import mean_squared_error

if __name__ == "__main__":
    model_path = f"/opt/ml/processing/model/model.tar.gz"
    with tarfile.open(model_path) as tar:
        tar.extractall(path=".")

```

```

model = pickle.load(open("xgboost-model", "rb"))

test_path = "/opt/ml/processing/test/test.csv"
df = pd.read_csv(test_path, header=None)

y_test = df.iloc[:, 0].to_numpy()
df.drop(df.columns[0], axis=1, inplace=True)

X_test = xgboost.DMatrix(df.values)

predictions = model.predict(X_test)

mse = mean_squared_error(y_test, predictions)
std = np.std(y_test - predictions)
report_dict = {
    "regression_metrics": {
        "mse": {
            "value": mse,
            "standard_deviation": std
        },
    },
}

output_dir = "/opt/ml/processing/evaluation"
pathlib.Path(output_dir).mkdir(parents=True, exist_ok=True)

evaluation_path = f"{output_dir}/evaluation.json"
with open(evaluation_path, "w") as f:
    f.write(json.dumps(report_dict))

```

2. Create an instance of a `ScriptProcessor` that is used to create a `ProcessingStep`.

```

from sagemaker.processing import ScriptProcessor

script_eval = ScriptProcessor(
    image_uri=image_uri,
    command=["python3"],
    instance_type=processing_instance_type,
    instance_count=1,
    base_job_name="script-abalone-eval",
    role=role,
)

```

3. Create a `ProcessingStep` using the processor instance, the input and output channels, and the `evaluation.py` script. In particular, pass in the `S3ModelArtifacts` property from the `step_train` training step, as well as the `S3Uri` of the "test" output channel of the `step_process` processing step. This is very similar to a processor instance's `run` method in the SageMaker Python SDK.

```

from sagemaker.workflow.properties import PropertyFile

evaluation_report = PropertyFile(
    name="EvaluationReport",
    output_name="evaluation",
    path="evaluation.json"
)
step_eval = ProcessingStep(
    name="AbaloneEval",
    processor=script_eval,
    inputs=[
        ProcessingInput(
            source=step_train.properties.ModelArtifacts.S3ModelArtifacts,

```

```

        destination="/opt/ml/processing/model"
    ),
    ProcessingInput(
        source=step_process.properties.ProcessingOutputConfig.Outputs[
            "test"
        ].S3Output.S3Uri,
        destination="/opt/ml/processing/test"
    )
],
outputs=[
    ProcessingOutput(output_name="evaluation", source="/opt/ml/processing/
evaluation"),
],
code="abalone/evaluation.py",
property_files=[evaluation_report],
)
)

```

## Step 6: Define a CreateModelStep for Batch Transformation

This section shows how to create a SageMaker model from the output of the training step. This model is used for batch transformation on a new dataset. This step is passed into the condition step and only executes if the condition step evaluates to true.

### To define a CreateModelStep for batch transformation

1. Create a SageMaker model. Pass in the `S3ModelArtifacts` property from the `step_train` training step.

```

from sagemaker.model import Model

model = Model(
    image_uri=image_uri,
    model_data=step_train.properties.ModelArtifacts.S3ModelArtifacts,
    sagemaker_session=sagemaker_session,
    role=role,
)

```

2. Define the model input for your SageMaker model.

```

from sagemaker.inputs import CreateModelInput

inputs = CreateModelInput(
    instance_type="ml.m5.large",
    accelerator_type="ml.eia1.medium",
)

```

3. Create your `CreateModelStep` using the `CreateModelInput` and SageMaker model instance you defined.

```

from sagemaker.workflow.steps import CreateModelStep

step_create_model = CreateModelStep(
    name="AbaloneCreateModel",
    model=model,
    inputs=inputs,
)

```

## Step 7: Define a TransformStep to Perform Batch Transformation

This section shows how to create a `TransformStep` to perform batch transformation on a dataset after the model is trained. This step is passed into the condition step and only executes if the condition step evaluates to true.

### To define a `TransformStep` to perform batch transformation

1. Create a transformer instance with the appropriate compute instance type, instance count, and desired output Amazon S3 bucket URI. Pass in the `ModelName` property from the `step_create_model` `CreateModel` step.

```
from sagemaker.transformer import Transformer

transformer = Transformer(
    model_name=step_create_model.properties.ModelName,
    instance_type="ml.m5.xlarge",
    instance_count=1,
    output_path=f"s3://{default_bucket}/AbaloneTransform"
)
```

2. Create a `TransformStep` using the transformer instance you defined and the `batch_data` pipeline parameter.

```
from sagemaker.inputs import TransformInput
from sagemaker.workflow.steps import TransformStep

step_transform = TransformStep(
    name="AbaloneTransform",
    transformer=transformer,
    inputs=TransformInput(data=batch_data)
)
```

## Step 8: Define a RegisterModel Step to Create a Model Package

This section shows how to construct an instance of `RegisterModel`. The result of executing `RegisterModel` in a pipeline is a model package. A model package is a reusable model artifacts abstraction that packages all ingredients necessary for inference. It consists of an inference specification that defines the inference image to use along with an optional model weights location. A model package group is a collection of model packages. You can use a `ModelPackageGroup` for SageMaker Pipelines to add a new version and model package to the group for every pipeline execution. For more information about model registry, see [Register and Deploy Models with Model Registry \(p. 2003\)](#).

This step is passed into the condition step and only executes if the condition step evaluates to true.

### To define a `RegisterModel` step to create a model package

- Construct a `RegisterModel` step using the estimator instance you used for the training step . Pass in the `S3ModelArtifacts` property from the `step_train` training step and specify a `ModelPackageGroup`. SageMaker Pipelines creates this `ModelPackageGroup` for you.

```
from sagemaker.model_metrics import MetricsSource, ModelMetrics
from sagemaker.workflow.step_collections import RegisterModel

model_metrics = ModelMetrics(
    model_statistics=MetricsSource(
```

```

        s3_uri="{}/evaluation.json".format(
            step_eval.arguments["ProcessingOutputConfig"]["Outputs"][0]["S3Output"]
        ["S3Uri"]
        ),
        content_type="application/json"
    )
)
step_register = RegisterModel(
    name="AbaloneRegisterModel",
    estimator=xgb_train,
    model_data=step_train.properties.ModelArtifacts.S3ModelArtifacts,
    content_types=["text/csv"],
    response_types=["text/csv"],
    inference_instances=["ml.t2.medium", "ml.m5.xlarge"],
    transform_instances=["ml.m5.xlarge"],
    model_package_group_name=model_package_group_name,
    approval_status=model_approval_status,
    model_metrics=model_metrics
)

```

## Step 9: Define a Condition Step to Verify Model Accuracy

A ConditionStep allows SageMaker Pipelines to support conditional execution in your pipeline DAG based on the condition of step properties. In this case, you only want to register a model package if the accuracy of that model, as determined by the model evaluation step, exceeds the required value. If the accuracy exceeds the required value, the pipeline also creates a SageMaker Model and runs batch transformation on a dataset. This section shows how to define the Condition step.

### To define a condition step to verify model accuracy

1. Define a ConditionLessThanOrEqualTo condition using the accuracy value found in the output of the model evaluation processing step, step\_eval. Get this output using the property file you indexed in the processing step and the respective JSONPath of the mean squared error value, "mse".

```

from sagemaker.workflow.conditions import ConditionLessThanOrEqualTo
from sagemaker.workflow.condition_step import (
    ConditionStep,
    JsonGet,
)

cond_lte = ConditionLessThanOrEqualTo(
    left=JsonGet(
        step=step_eval,
        property_file=evaluation_report,
        json_path="regression_metrics.mse.value"
    ),
    right=6.0
)

```

2. Construct a ConditionStep. Pass the ConditionEquals condition in, then set the model package registration and batch transformation steps as the next steps if the condition passes.

```

step_cond = ConditionStep(
    name="AbaloneMSECond",
    conditions=[cond_lte],
    if_steps=[step_register, step_create_model, step_transform],
    else_steps=[],
)

```

## Step 10: Create a pipeline

Now that you've created all of the steps, combine them into a pipeline.

### To create a pipeline

1. Define the following for your pipeline: `name`, `parameters`, and `steps`. Names must be unique within an `(account, region)` pair.

**Note**

A step can only appear once in either the pipeline's step list or the if/else step lists of the condition step. It cannot appear in both.

```
from sagemaker.workflow.pipeline import Pipeline

pipeline_name = f"AbalonePipeline"
pipeline = Pipeline(
    name=pipeline_name,
    parameters=[
        processing_instance_type,
        processing_instance_count,
        training_instance_type,
        model_approval_status,
        input_data,
        batch_data,
    ],
    steps=[step_process, step_train, step_eval, step_cond],
)
```

2. (Optional) Examine the JSON pipeline definition to ensure that it's well-formed.

```
import json

json.loads(pipeline.definition())
```

This pipeline definition is ready to submit to SageMaker. In the next tutorial, you submit this pipeline to SageMaker and start an execution.

**Next step:** [Run a pipeline \(p. 2214\)](#)

## Run a pipeline

After you've created a pipeline definition using the SageMaker Python SDK, you can submit it to SageMaker to start your execution. The following tutorial shows how to submit a pipeline, start an execution, examine the results of that execution, and delete your pipeline.

### Topics

- [Prerequisites \(p. 2214\)](#)
- [Step 1: Start the Pipeline \(p. 2215\)](#)
- [Step 2: Examine a Pipeline Execution \(p. 2215\)](#)
- [Step 3: Override Default Parameters for a Pipeline Execution \(p. 2216\)](#)
- [Step 4: Stop and Delete a Pipeline Execution \(p. 2217\)](#)

## Prerequisites

This tutorial requires the following:

- A SageMaker notebook instance.
- A SageMaker Pipelines pipeline definition. This tutorial assumes you're using the pipeline definition created by completing the [Define a Pipeline \(p. 2201\)](#) tutorial.

## Step 1: Start the Pipeline

First, you need to start the pipeline.

### To start the pipeline

1. Examine the JSON pipeline definition to ensure that it's well-formed.

```
import json
json.loads(pipeline.definition())
```

2. Submit the pipeline definition to the SageMaker Pipelines service to create a pipeline if it doesn't exist, or update the pipeline if it does. The role passed in is used by SageMaker Pipelines to create all of the jobs defined in the steps.

```
pipeline.upsert(role_arn=role)
```

3. Start a pipeline execution.

```
execution = pipeline.start()
```

## Step 2: Examine a Pipeline Execution

Next, you need to examine the pipeline execution.

### To examine a pipeline execution

1. Describe the pipeline execution status to ensure that it has been created and started successfully.

```
execution.describe()
```

2. Wait for the execution to finish.

```
execution.wait()
```

3. List the execution steps and their status.

```
execution.list_steps()
```

Your output should look like the following:

```
[{'StepName': 'AbaloneTransform',
  'StartTime': datetime.datetime(2020, 11, 21, 2, 41, 27, 870000, tzinfo=tzlocal()),
  'EndTime': datetime.datetime(2020, 11, 21, 2, 45, 50, 492000, tzinfo=tzlocal()),
  'StepStatus': 'Succeeded',
  'CacheHitResult': {'SourcePipelineExecutionArn': ''},
  'Metadata': {'TransformJob': {'Arn': 'arn:aws:sagemaker:us-
east-2:111122223333:transform-job/pipelines-cfvyltjuxdq8-abalonetransform-
ptyjoef3jy'}},
  {'StepName': 'AbaloneRegisterModel',
  'StartTime': datetime.datetime(2020, 11, 21, 2, 41, 26, 929000, tzinfo=tzlocal()),
```

```

'EndTime': datetime.datetime(2020, 11, 21, 2, 41, 28, 15000, tzinfo=tzlocal()),
'StepStatus': 'Succeeded',
'CacheHitResult': {'SourcePipelineExecutionArn': ''},
'Metadata': {'RegisterModel': {'Arn': 'arn:aws:sagemaker:us-
east-2:111122223333:model-package/abalonemodelpackagegroupname/1'}},
{'StepName': 'AbaloneCreateModel',
 'StartTime': datetime.datetime(2020, 11, 21, 2, 41, 26, 895000, tzinfo=tzlocal()),
 'EndTime': datetime.datetime(2020, 11, 21, 2, 41, 27, 708000, tzinfo=tzlocal()),
 'StepStatus': 'Succeeded',
 'CacheHitResult': {'SourcePipelineExecutionArn': ''},
 'Metadata': {'Model': {'Arn': 'arn:aws:sagemaker:us-east-2:111122223333:model/
pipelines-cfvyltjuxdq8-abalonecreatemodel-jl94rai0ra'}},
{'StepName': 'AbaloneMSECond',
 'StartTime': datetime.datetime(2020, 11, 21, 2, 41, 25, 558000, tzinfo=tzlocal()),
 'EndTime': datetime.datetime(2020, 11, 21, 2, 41, 26, 329000, tzinfo=tzlocal()),
 'StepStatus': 'Succeeded',
 'CacheHitResult': {'SourcePipelineExecutionArn': ''},
 'Metadata': {'Condition': {'Outcome': 'True'}},
{'StepName': 'AbaloneEval',
 'StartTime': datetime.datetime(2020, 11, 21, 2, 37, 34, 767000, tzinfo=tzlocal()),
 'EndTime': datetime.datetime(2020, 11, 21, 2, 41, 18, 80000, tzinfo=tzlocal()),
 'StepStatus': 'Succeeded',
 'CacheHitResult': {'SourcePipelineExecutionArn': ''},
 'Metadata': {'ProcessingJob': {'Arn': 'arn:aws:sagemaker:us-
east-2:111122223333:processing-job/pipelines-cfvyltjuxdq8-abaloneeval-zfrazohmny'}},
{'StepName': 'AbaloneTrain',
 'StartTime': datetime.datetime(2020, 11, 21, 2, 34, 55, 867000, tzinfo=tzlocal()),
 'EndTime': datetime.datetime(2020, 11, 21, 2, 37, 34, 34000, tzinfo=tzlocal()),
 'StepStatus': 'Succeeded',
 'CacheHitResult': {'SourcePipelineExecutionArn': ''},
 'Metadata': {'TrainingJob': {'Arn': 'arn:aws:sagemaker:us-
east-2:111122223333:training-job/pipelines-cfvyltjuxdq8-abalonetrain-tavd6f3wdf'}},
{'StepName': 'AbaloneProcess',
 'StartTime': datetime.datetime(2020, 11, 21, 2, 30, 27, 160000, tzinfo=tzlocal()),
 'EndTime': datetime.datetime(2020, 11, 21, 2, 34, 48, 390000, tzinfo=tzlocal()),
 'StepStatus': 'Succeeded',
 'CacheHitResult': {'SourcePipelineExecutionArn': ''},
 'Metadata': {'ProcessingJob': {'Arn': 'arn:aws:sagemaker:us-
east-2:111122223333:processing-job/pipelines-cfvyltjuxdq8-abaloneprocess-
mgqyfdujcj'}}}]

```

- After your pipeline execution is complete, download the resulting `evaluation.json` file from Amazon S3 to examine the report.

```

evaluation_json = sagemaker.s3.S3Downloader.read_file("{}/evaluation.json".format(
    step_eval.arguments["ProcessingOutputConfig"]["Outputs"][0]["S3Output"]["S3Uri"]
))
json.loads(evaluation_json)

```

## Step 3: Override Default Parameters for a Pipeline Execution

You can run additional executions of the pipeline by specifying different pipeline parameters to override the defaults.

### To override default parameters

- Create the pipeline execution. This starts another pipeline execution on a compute-optimized instance type and sets the model approval status to "Approved". This means that the model package version generated by the `RegisterModel` step is automatically ready for deployment through CI/CD pipelines, such as with SageMaker Projects. For more information, see [Automate MLOps with SageMaker Projects \(p. 2232\)](#).

```
execution = pipeline.start(  
    parameters=dict(  
        ProcessingInstanceType="ml.c5.xlarge",  
        ModelApprovalStatus="Approved",  
    )  
)
```

2. Wait for the execution to finish.

```
execution.wait()
```

3. List the execution steps and their status.

```
execution.list_steps()
```

4. After your pipeline execution is complete, download the resulting `evaluation.json` file from Amazon S3 to examine the report.

```
evaluation_json = sagemaker.s3.S3Downloader.read_file("{}/evaluation.json".format(  
    step_eval.arguments["ProcessingOutputConfig"]["Outputs"][0]["S3Output"]["S3Uri"]  
)  
json.loads(evaluation_json)
```

## Step 4: Stop and Delete a Pipeline Execution

When you're finished with your pipeline, you can stop any ongoing executions and delete the pipeline.

### To stop and delete a pipeline execution

1. Stop the pipeline execution.

```
execution.stop()
```

2. Delete the pipeline.

```
pipeline.delete()
```

## View, Track, and Execute SageMaker Pipelines in SageMaker Studio

To view, track, and execute Amazon SageMaker Pipelines in Amazon SageMaker Studio, you must sign in to Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).

### Topics

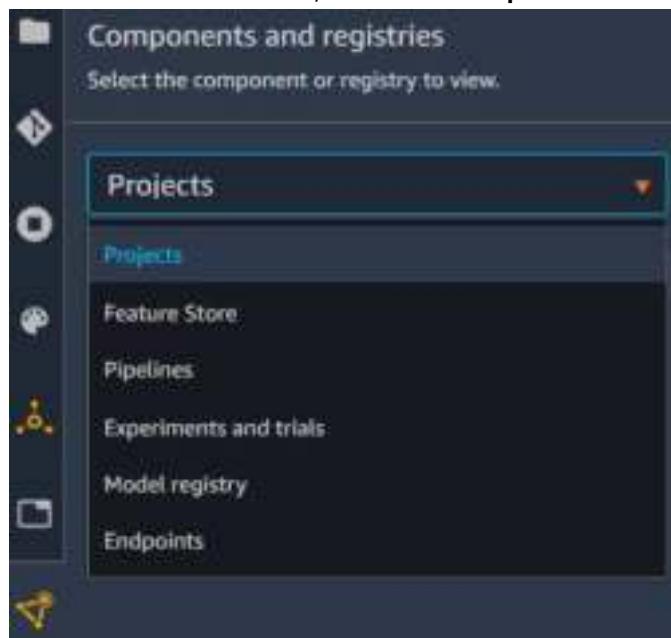
- [View a Pipeline \(p. 2218\)](#)
- [View a Pipeline Execution \(p. 2221\)](#)
- [View Experiment Entities Created by SageMaker Pipelines \(p. 2224\)](#)
- [Execute a Pipeline \(p. 2226\)](#)
- [Track the Lineage of a SageMaker ML Pipeline \(p. 2228\)](#)

## View a Pipeline

This procedure shows you how to directly find a pipeline and view its details page. You can also find pipelines that are part of a project listed in the project's details page. For information on finding a pipeline that is part of a project, see [Automate MLOps with SageMaker Projects \(p. 2232\)](#).

### To view a list of pipelines

1. In the left sidebar of Studio, choose the **Components and registries** icon (💡).



2. Select **Pipelines** from the dropdown list.

Pipelines	
<input type="text"/> Search column name to start	
Name	Last modified
AbalonePipeline	2 days ago
CensusProcessTrainEvalC...	11 days ago
AbaloneProcessTrainEval...	11 days ago
AbaloneProcessTrainEval...	18 days ago

3. Drag the right border of the **Components and registries** pane to the right to view all the columns. Use search to narrow the list of pipelines.

Search is a two-step process. First, you enter characters that match the column name you want to search. Second, you enter characters to match the item in that column.

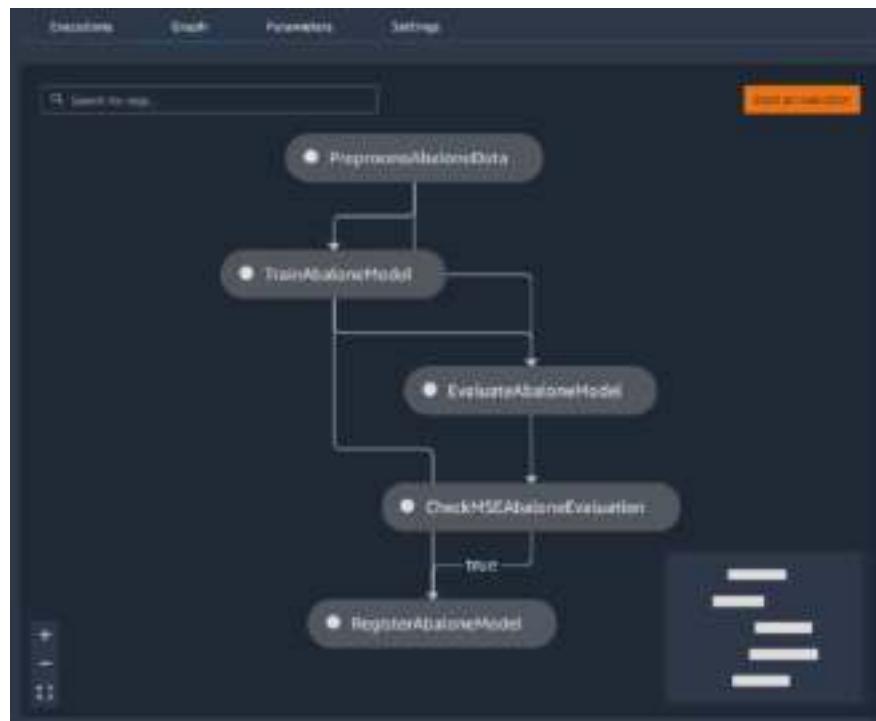
You can have multiple search filters. As an example, the following screenshot limits the displayed pipelines to those with a name that starts with aba and created by Me. For more information on searching in Studio, see [Search Experiments Using Amazon SageMaker Studio \(p. 1567\)](#).

The screenshot shows the 'Components and registries' section of the Amazon SageMaker Studio interface. A search bar at the top says 'Select the component or registry to view.' Below it, a dropdown menu is set to 'Pipelines'. A search bar labeled 'Search column name to start' contains 'Name: aba'. Below the search bar are three buttons: 'Name: aba' with an 'X', 'Created by: Me' with an 'X', and 'Clear all'. A table follows, with columns: Name, Created, Created by, and Last modified. One row is visible: 'AbaloneProcessTrainEval...' was created '7 days ago' by 'kudzu-1' and last modified '7 days ago'. At the bottom of the table, it says 'End of the list'.

4. Open a pipeline to view details about the pipeline. The pipeline details tab opens and displays a list of pipeline executions. You can start an execution or choose one of the other tabs for more information about the pipeline. Use the **Settings** icon ( ) to choose which columns to display.

The screenshot shows the 'AbaloneProcessTrainEvalCondRegister-16050297567595590' pipeline details page. It features tabs for 'Overview', 'Graph', 'Execution', and 'Settings'. The 'Execution' tab is active, showing a table of pipeline executions. The table has columns: Status, Step, Pipeline, Last modified, and Last run. One execution is listed: 'Completed' status, 'Step 1' step, 'AbaloneProcessTrainEval...' pipeline, '7 days ago' last modified, and 'Completed' last run. To the right of the table is a 'TABLE PREFERENCES' sidebar with options like 'Auto-refresh interval', 'Execute group parallel', and a list of columns to include/exclude. An orange 'Start a new exec' button is located to the right of the table.

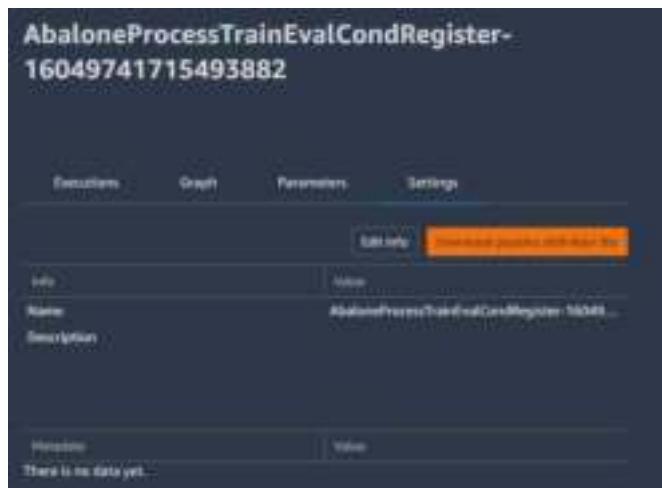
5. From the pipeline details page, choose one of the following tabs to view details about the pipeline:
  - **Executions** – Details about the executions. You can start an execution from this tab or the **Graph** tab.
  - **Graph** – The DAG for the pipeline.



- **Parameters** – Includes the model approval status.

Name	Type	Value
ProcessingInstanceType	String	m4.4xlarge
ProcessingInstanceCount	Integer	1
TrainingInstanceType	String	m4.4xlarge
ModelApprovalStatus	String	PendingManualApproval
InputData	String	<a href="https://s3-us-west-2.amazonaws.com">https://s3-us-west-2.amazonaws.com</a>

- **Settings** – The metadata associated with the pipeline. You can download the pipeline definition file and edit the pipeline name and description from this tab.



## View a Pipeline Execution

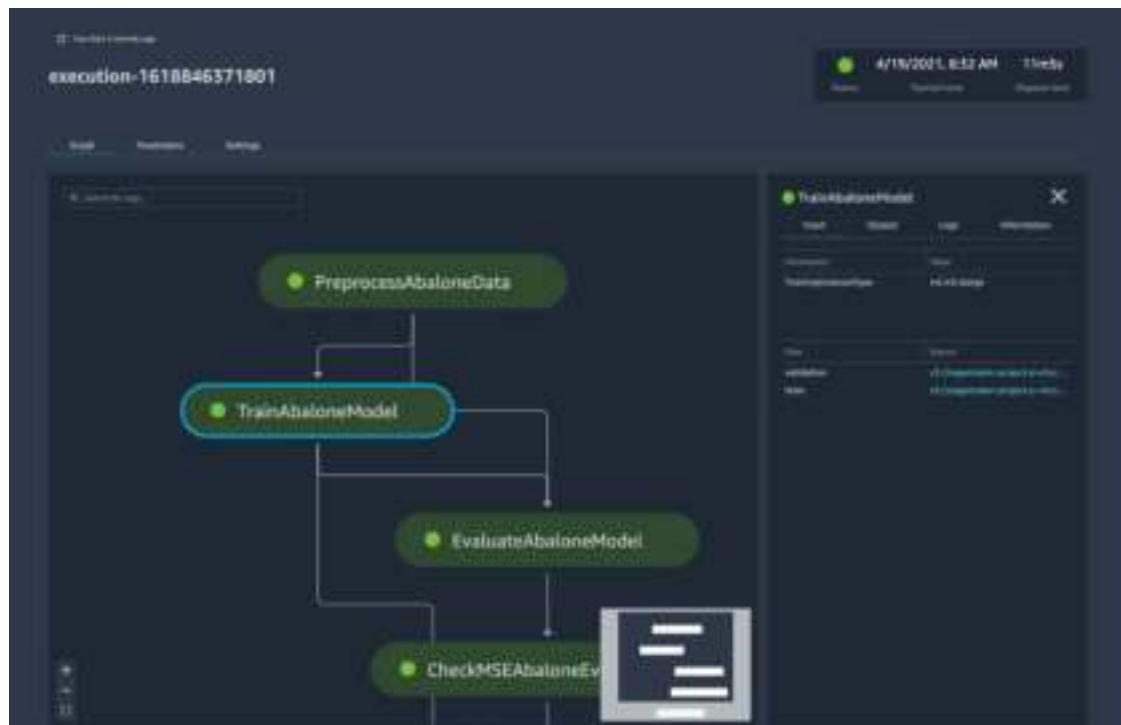
This procedure shows you how to view a pipeline execution. For information on how to view a list of pipeline executions, and how to use SageMaker search to narrow the executions in the list, see [View a Pipeline \(p. 2218\)](#).

### To view details of a pipeline execution

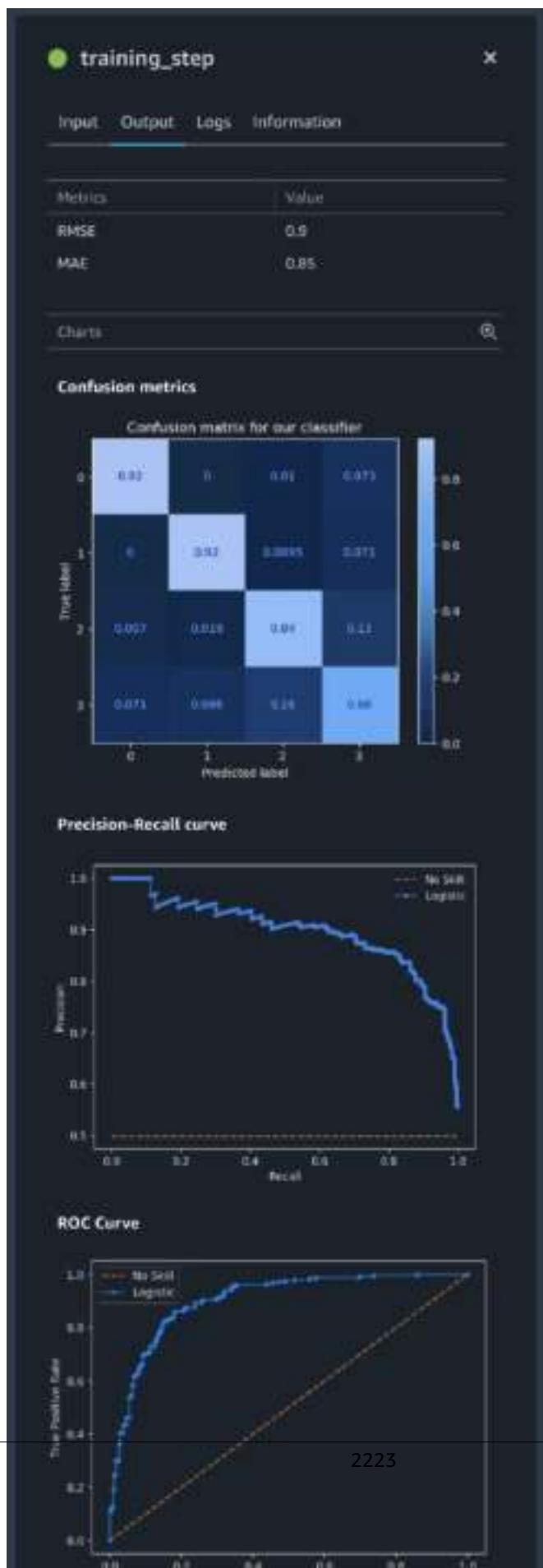
1. In the execution list, open an execution to view details about the execution. The execution details tab opens and displays a graph of the steps in the pipeline. You can choose one of the other tabs to view information about the pipeline execution, which is similar to that shown when viewing the pipeline details.



2. Use search to find a step in the graph. Type characters that match a step name. You can drag the graph around or use the resizing icons on the lower-left side of the graph. The inset on the lower-right side of the graph displays where you are in the graph.



3. Choose one of the steps in the graph to see details about the step. In the preceding screenshot, a training step is chosen and displays the following tabs:
  - **Input** – The training inputs. If an input source is from Amazon Simple Storage Service (Amazon S3), choose the link to view the file in the Amazon S3 console.
  - **Output** – The training outputs, such as metrics, charts, files, and evaluation outcome. The graphs were produced using the [Tracker APIs](#).



- **Logs** – The Amazon CloudWatch logs produced by the step.

Output	Logs	Info
<a href="#">View logs in CloudWatch console</a>		
CW logs		
	1605069433822   [ 2020-11-11T04:57:13.822Z ] [49]#011train-rmse:1.59950#011validation-rmse:2.18827	
	1605069433822   [ 2020-11-11T04:57:13.822Z ] [48]#011train-rmse:1.60384#011validation-rmse:2.18565	
	1605069433822   [ 2020-11-11T04:57:13.822Z ] [47]#011train-rmse:1.61095#011validation-rmse:2.17939	
	1605069433822   [ 2020-11-11T04:57:13.822Z ] [46]#011train-rmse:1.61517#011validation-rmse:2.17707	

- **Info** – The parameters and metadata associated with the step.

Output	Logs	Info
<a href="#">View logs in CloudWatch console</a>		
Parameter		
This node has no parameters.		
<a href="#">View details</a>		
<a href="#">Edit</a>		
<a href="#">Delete</a>		
Metadata	Value	
Arn	arn:aws:sagemaker:us-east-2:...	

## View Experiment Entities Created by SageMaker Pipelines

When you create a pipeline and specify `pipeline_experiment_config`, SageMaker Pipelines creates the following SageMaker Experiments entities by default if they don't exist:

- An experiment for the pipeline
- A trial for every execution of the pipeline
- A trial component for each SageMaker job created in a pipeline step

For information on how experiments are integrated with pipelines, see [Amazon SageMaker Experiments Integration \(p. 2195\)](#). For more information on SageMaker Experiments, see [Manage Machine Learning with Amazon SageMaker Experiments \(p. 1553\)](#).

You can get to the list of trial components associated with a pipeline from either the pipeline executions list or the experiments list.

#### To view the trial components list from the pipeline executions list

1. To view the pipeline executions list, follow the first four steps in [View a Pipeline \(p. 2218\)](#).
2. On the top right of the screen, choose the **Settings** icon (⚙️) to open **TABLE PROPERTIES**.
3. Select **Experiment**. If experiment integration wasn't disabled when the pipeline was created, the experiment name and trial name are displayed in the executions list.

##### Note

Experiments integration was introduced in v2.41.0 of the [Amazon SageMaker Python SDK](#). Pipelines created with an earlier version of the SDK aren't integrated with experiments by default.

4. Select one or more executions, open (right-click) the selection, then choose **View trial components generated by execution**.

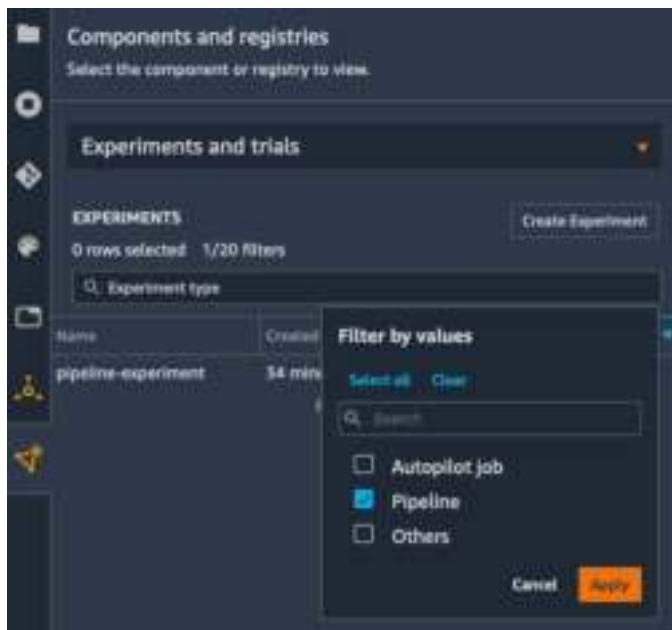


#### To view the trial components list from the experiments list

1. In the left sidebar of Studio, choose the **Components and registries** icon (💡).
2. Select **Experiments and trials** from the dropdown list.
3. Use search to filter the list to experiments created by a pipeline.
  - a. Drag the right border of the **Components and registries** pane to the right until you can see the **Experiment type** column.
  - b. In the search box, enter ex and choose **Experiment type**.
  - c. In **Filter by values**, choose **Pipeline**, then choose **Apply**.

To further narrow the search, you can add additional filters. For more information, see [Search Experiments Using Amazon SageMaker Studio \(p. 1567\)](#).

4. Open (right-click) an experiment name and choose **Open in trial component list**.



### List of trial components

The following shows the list of trial components created by the pipeline.

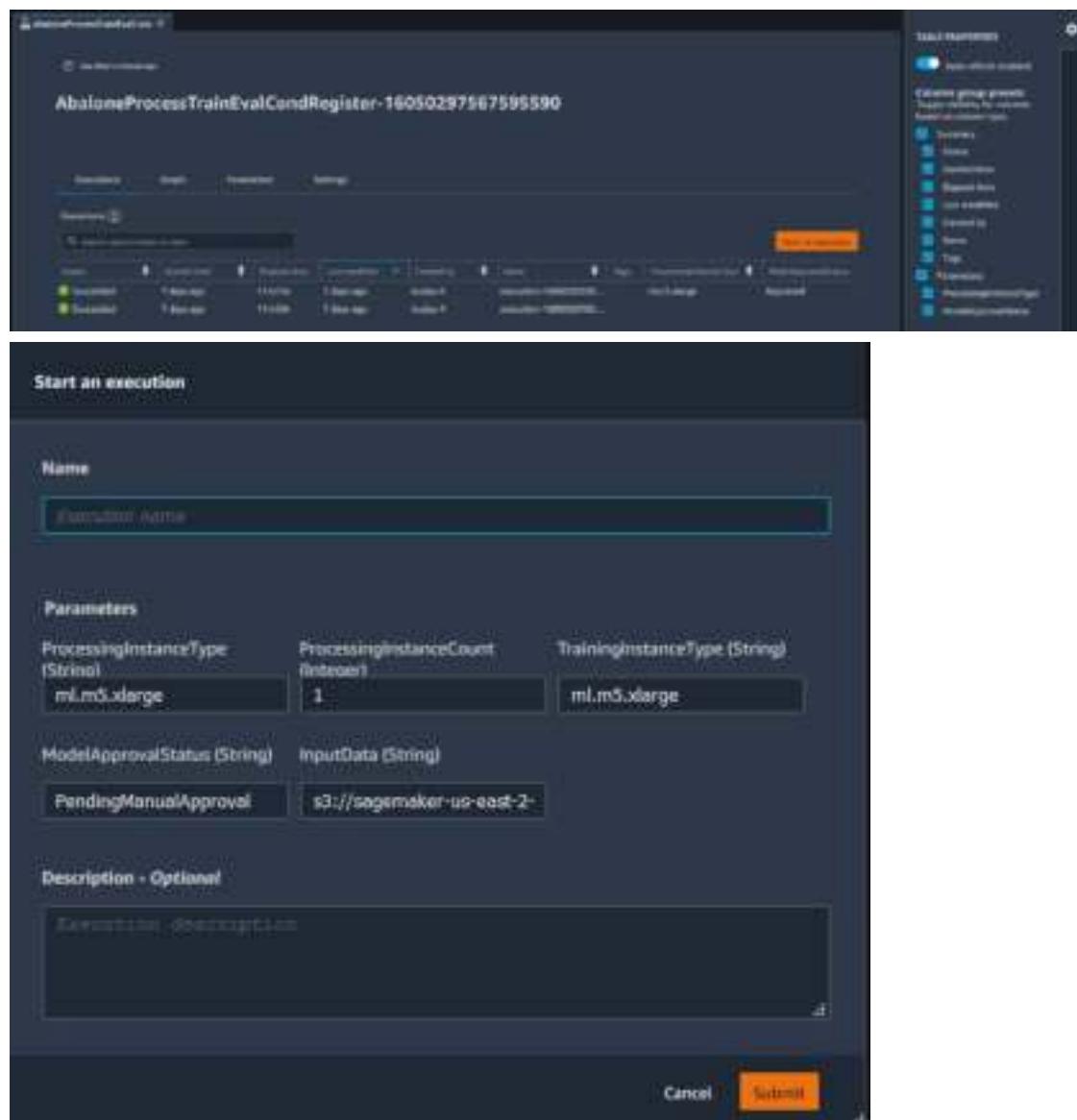
Trial Component List							
Status		Experiment name	Trial name	Trial component name	Trial component type	Created	Last modified
Completed	pipeline-experiment	pipeline-experiment	pipeline-trial	pipeline-step1-step47	Processing job	24 minutes ago	19 minutes ago
Completed	pipeline-experiment	pipeline-experiment	pipeline-trial	pipeline-step1-step47	Training job	25 minutes ago	23 minutes ago
Completed	pipeline-experiment	pipeline-experiment	pipeline-trial	pipeline-step1-step47	Processing job	25 minutes ago	23 minutes ago

### Execute a Pipeline

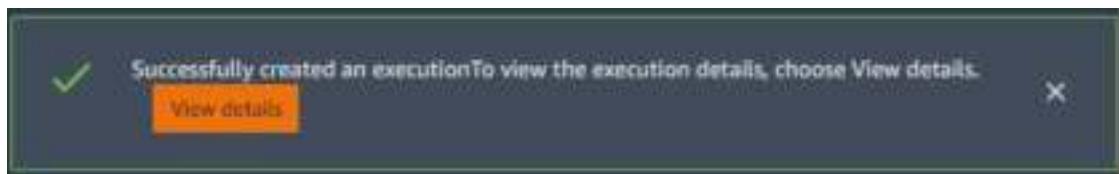
This procedure shows you how to execute a pipeline. For information on how to view a list of pipeline executions, see [View a Pipeline \(p. 2218\)](#).

#### To start a pipeline execution

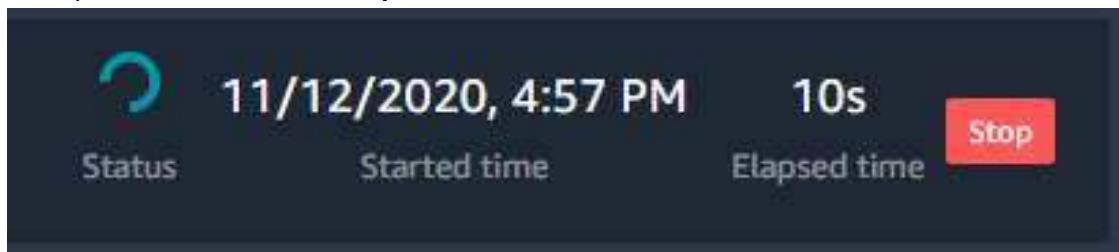
1. From the **Executions** or **Graph** tab in the execution list, choose **Start an execution**.



2. Enter or update the following required information:
  - **Name** – Must be unique to your account in the Amazon Region.
  - **ProcessingInstanceType** – The instance type for processing.
  - **ProcessingInstanceCount** – The number of instances to use for processing.
  - **TrainingInstanceType** – The instance type for training.
  - **ModelApprovalStatus** – For your convenience.
  - **InputData** – The S3 URI of the input data.
3. Choose **Submit**.
4. To see details of the execution or to stop the execution, choose **View details** on the status banner.



5. To stop the execution, choose **Stop** on the status banner.



To view the list of registered models, see [Automate MLOps with SageMaker Projects \(p. 2232\)](#).

## Track the Lineage of a SageMaker ML Pipeline

In this tutorial, you use Amazon SageMaker Studio to track the lineage of an Amazon SageMaker ML Pipeline.

The pipeline was created by the [Orchestrating Jobs with Amazon SageMaker Model Building Pipelines](#) notebook in the [Amazon SageMaker example GitHub repository](#). For detailed information on how the pipeline was created, see [Define a Pipeline \(p. 2201\)](#).

Lineage tracking in Studio is centered around a directed acyclic graph (DAG). The DAG represents the steps in a pipeline. From the DAG you can track the lineage from any step to any other step. The following diagram displays the steps in the pipeline. These steps appear as a DAG in Studio.

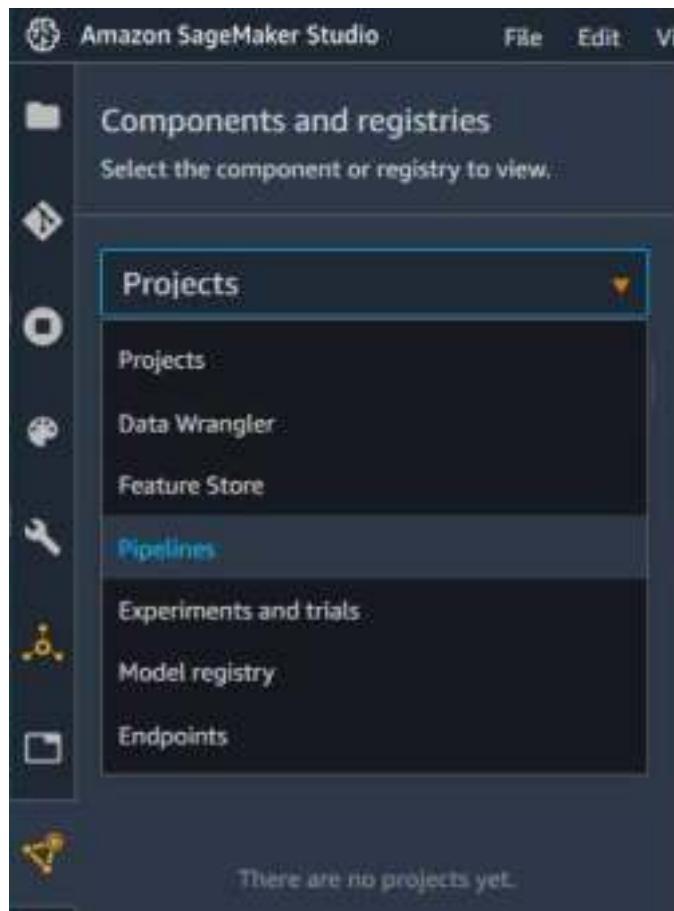


### Prerequisites

- Access to Amazon SageMaker Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
- Familiarity with the SageMaker Studio user interface. For more information, see [Amazon SageMaker Studio UI Overview \(p. 69\)](#).
- (Recommended) A completed run of the example notebook.

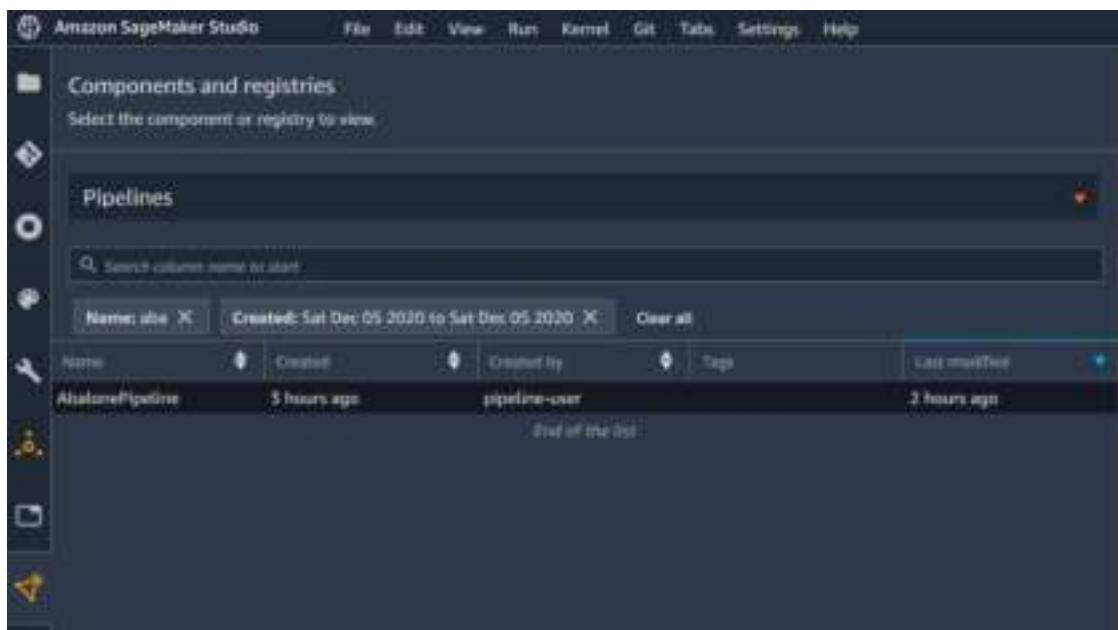
### To track the lineage of a pipeline

1. Sign in to SageMaker Studio.
2. In the left sidebar of Studio, choose the **SageMaker Components and registries** icon (💡).
3. In the drop-down menu, select **Pipelines**.



4. Use the **Search** box to filter the pipelines list. To view all available columns, drag the right border of the pane to the right. For more information, see [Search Experiments Using Amazon SageMaker Studio \(p. 1567\)](#).

The following screenshot shows the list filtered by a name that starts with "aba" and that was created on 12/5/20.

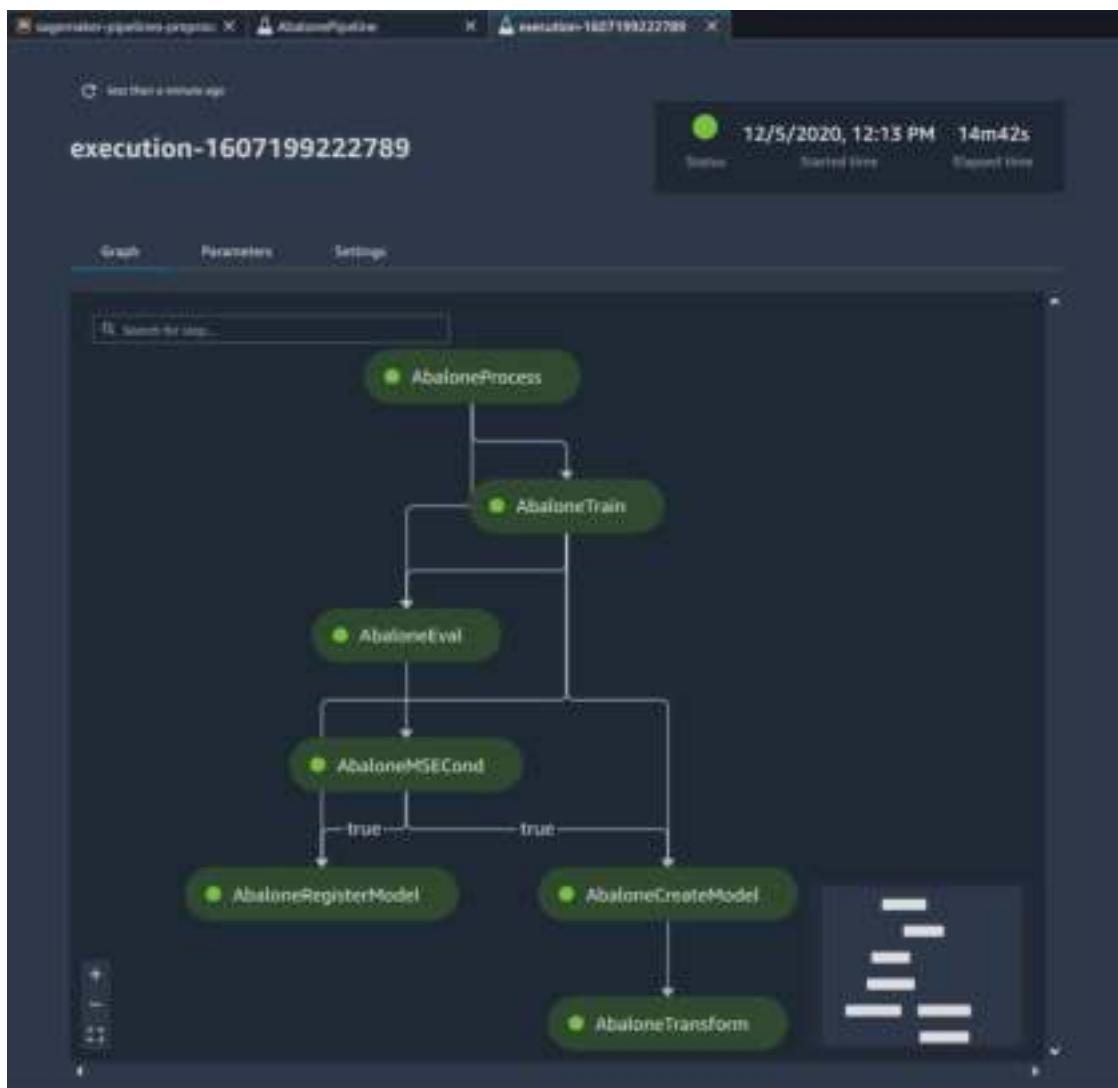


- Double-click the AbalonePipeline pipeline to view the execution list and other details about the pipeline. The following screenshot shows the **TABLE PROPERTIES** pane open where you can choose which properties to view.



- Choose the **Settings** tab and then choose **Download pipeline definition file**. You can view the file to see how the pipeline graph was defined.
- On the **Execution** tab, double-click the first row in the execution list to view its execution graph and other details about the execution. Note that the graph matches the diagram displayed at the beginning of the tutorial.

You can drag the graph around (select an area not on the graph itself) or use the resizing icons on the lower-left side of the graph. The inset on the lower-right side of the graph displays your location in the graph.



8. On the **Graph** tab, choose the **AbaloneProcess** step to view details about the step.



9. Find the Amazon S3 paths to the training, validation, and test datasets in the **Output** tab, under **Files**.

**Note**

To get the full paths, right-click the path and then choose **Copy cell contents**.

```
s3://sagemaker-eu-west-1-acct-id/sklearn-abalone-process-2020-12-05-17-28-28-509/
output/train
s3://sagemaker-eu-west-1-acct-id/sklearn-abalone-process-2020-12-05-17-28-28-509/
output/validation
s3://sagemaker-eu-west-1-acct-id/sklearn-abalone-process-2020-12-05-17-28-28-509/
output/test
```

10. Choose the AbaloneTrain step.



11. Find the Amazon S3 path to the model artifact in the **Output** tab, under **Files**:

```
s3://sagemaker-eu-west-1-acct-id/AbaloneTrain/pipelines-6locnsqz4bfu-AbaloneTrain-NtfEpIOAhu/output/model.tar.gz
```

12. Choose the AbaloneRegisterModel step.



13. Find the ARN of the model package in the **Output** tab, under **Files**:

```
arn:aws:sagemaker:eu-west-1:acct-id:model-package/abalonemodelpackagegroupname/2
```

## Automate MLOps with SageMaker Projects

Create end-to-end ML solutions with CI/CD by using SageMaker projects.

Use SageMaker projects to create an ML ops solution to orchestrate and manage:

- Data preparation and feature engineering
- Training models
- Evaluating models
- Deploying models

- Monitor and update models

#### Topics

- [What is a SageMaker Project? \(p. 2233\)](#)
- [Why Should You Use MLOps? \(p. 2234\)](#)
- [SageMaker Studio Permissions Required to Use Projects \(p. 2236\)](#)
- [Create an MLOps Project \(p. 2236\)](#)
- [MLOps Project Templates \(p. 2240\)](#)
- [View Project Resources \(p. 2242\)](#)
- [SageMaker MLOps Project Walkthrough \(p. 2244\)](#)

## What is a SageMaker Project?

By using a SageMaker project, teams of data scientists and developers can work in machine learning business problems. You can create a SageMaker project with a SageMaker provided MLOPs template that automates the model building and deployment pipelines using continuous integrations and continuous delivery (CI/CD). A SageMaker provided template provisions the initial setup required for a complete end-to-end MLOps system including model building, training, and deployment, depending on the template you choose. You can also provide your own custom template to provision the resources you need for your MLOps system.

A SageMaker project is an Amazon Service Catalog provisioned product that enables you to easily create an end-to-end ML solution. For information about Amazon Service Catalog, see [What is Amazon Service Catalog](#).

Each SageMaker project has a unique name and ID that are passed to all SageMaker and Amazon resources created in the project. By using the name and ID, you can view all entities associated with your project. These include:

- Pipeline executions
- Registered models
- Deployed models (endpoints)
- Datasets
- Amazon Service Catalog Products
- Amazon CodePipeline pipelines
- Amazon CodeCommit repositories

A typical SageMaker project with a CICD template might include the following.

- One or more CodeCommit repositories with sample code for building and deploying ML solutions. These are working examples that you can clone locally to explore the code provided by SageMaker and modify for your needs. You own this code and you can use the repositories as version control for your work.
- A SageMaker pipeline that defines steps for data preparation, training, model evaluation, and model deployment.
- A CodePipeline that runs your SageMaker pipeline every time a new version of the code is checked in. For information about CodePipeline, see [What is Amazon CodePipeline](#).
- A model group that contains model versions. Each time the SageMaker pipeline runs, and the resulting model version is accepted in the conditional validation step, a new model version is deployed to a SageMaker endpoint.

## When Should You Use a SageMaker Project?

While notebooks are helpful for model building and experimentation, when you have a team of data scientists and/or ML engineers working on an ML problem, you need a more scalable way to maintain code consistency and have stricter version control. Having the code only in notebook files makes it harder to collaborate and risks losing code or model artifacts if the notebook is accidentally deleted or changed. By using SageMaker projects, you can manage the versions for your Git repositories so you can collaborate across teams more efficiently, ensure code consistency, and enable CI/CD.

In addition to managing code, SageMaker projects enable MLOps for model building, model deployment, and end-to-end ML workflows. You can run training jobs or SageMaker pipelines to build models in SageMaker Studio. However, if you want to create a CI/CD system that generates models based on triggers, such as when someone checks in a code change, then consider creating a SageMaker project and using a SageMaker provided template. For a list of the project templates that SageMaker provides, see [Use SageMaker Provided Project Templates \(p. 2241\)](#).

## Do I Need to Create a Project to Use SageMaker Pipelines?

No. SageMaker pipelines are standalone entities just like training jobs, processing jobs, and other SageMaker jobs within SageMaker. You can create, update, and run pipelines directly within a notebook by using the SageMaker Python SDK without using a SageMaker project.

Projects provide an additional layer to help you organize your code and adopt operational best practices that you need for a production-quality system.

## Why Should You Use MLOps?

As you move from running individual artificial intelligence and machine learning (AI/ML) projects to using AI/ML to transform your business at scale, the discipline of ML Operations (MLOps) can help. MLOps accounts for the unique aspects of AI/ML projects in project management, CI/CD, and quality assurance, helping customers improve delivery time, reduce defects, and make data scientists more productive. MLOps refers to a methodology that is built on applying DevOps practices to machine learning workloads. You can review the [Introduction to DevOps on Amazon](#) white paper for a discussion of DevOps principles. [Practicing CI/CD on Amazon](#) and [Infrastructure as Code](#) go deeper into implementation using Amazon services.

Like DevOps, MLOps relies on a collaborative and streamlined approach to the machine learning development lifecycle where the intersection of people, process and technology are required to optimize the end-to-end activities required to develop, build, and operate machine learning workloads.

MLOps focuses on the intersection of data science and data engineering in combination with existing DevOps practices to streamline model delivery across the machine learning development lifecycle. MLOps is the discipline of integrating ML workloads into release management, CI/CD, and operations. MLOps requires the integration of software development, operations, data engineering, and data science.

## Challenges with MLOps

- **Project management** - ML projects involve data scientists, a relatively new role, and one not often integrated into cross-functional teams. These new team members often speak a very different technical language than product owners and software engineers, compounding the usual problem of translating business requirements into technical requirements.
- **Communication and collaboration** - Building visibility on ML projects and enabling collaboration across different stakeholders such as data engineers, data scientists, ML engineers, and DevOps.
- **Everything is code**

- Use of production data in development activities, longer experimentation lifecycles, dependencies on data pipelines, retraining deployment pipelines, and unique metrics in evaluating the performance of a model.
- Models often have a lifecycle independent of the applications and systems integrating with those models.
- The entire end-to-end system is reproducible through versioned code and artifacts. DevOps projects use Infrastructure-as-Code(IaC) and Configuration-as-Code(CaC) to build environments, and Pipelines-as-code to ensure consistent CI/CD patterns. The pipelines have to integrate with Big Data and ML training workflows. That often means that our pipeline is a combination of a traditional CI/CD tool and another workflow engine. There are important policy concerns for many ML projects, so our pipeline may also need to enforce those policies. Biased input data produces biased results, an increasing concern for business stakeholders.
- **CI/CD**
  - In MLOps, the source data is a first-class input, along with source code. That's why MLOps calls for versioning the source data and triggering pipeline runs when the source or inference data changes.
  - Pipelines must also version the ML models along with their inputs and other outputs, in order to provide for traceability.
  - Automated testing must include proper validation of the ML model, during build phases and when the model is in production.
  - Build phases may include model training and retraining, a time-consuming and resource-intensive process. Pipelines must be granular enough to only perform a full training cycle when the source data or ML code changes, not when related components change.
  - Because machine learning code is typically a small part of an overall solution, a deployment pipeline may also incorporate the additional steps required to package your model for consumption as an API by other applications and systems.
- **Monitoring and logging**
  - In the feature engineering and model training phases, we need to capture model training metrics as well as model experiments. Tuning an ML model requires manipulating the form of the input data as well as algorithm hyperparameters, and we need to systematically capture those experiments. Experiment tracking helps data scientists work more effectively and gives us a reproducible snapshot of their work.
  - Deployed ML models require monitoring of the data passed to the model for inference, along with the standard endpoint stability and performance metrics. The monitoring system must also capture the quality of model output, as evaluated by an appropriate ML metric.

## Benefits of MLOps

Adopting MLOps practices gives you faster time-to-market for ML projects by delivering the following benefits.

- **Productivity** - Providing self-service environments with access to curated data sets lets data engineers and data scientists move faster and waste less time with missing or invalid data.
- **Repeatability** - Automating all the steps in the MLDC helps you ensure a repeatable process, including how the model is trained, evaluated, versioned, and deployed.
- **Reliability** - Incorporating CI/CD practices allows for the ability to not only deploy quickly but with increased quality and consistency.
- **Auditability** - Versioning all inputs and outputs, from data science experiments to source data to trained model, means that we can demonstrate exactly how the model was built, and where it was deployed.
- **Data and model quality** - MLOps lets us enforce policies that guard against model bias and track changes to data statistical properties and model quality over time.

# SageMaker Studio Permissions Required to Use Projects

To enable users to view SageMaker provided project templates and create projects with those templates, enable **Projects** permissions for users when you onboard or update SageMaker Studio. There are 2 permissions to enable.

1. Enable **Projects** permissions for the SageMaker Studio administrator. This enables the SageMaker Studio administrator to view the SageMaker provided templates in the Amazon Service Catalog console. The administrator can see what other SageMaker Studio users create if you grant them permission to use SageMaker projects. The administrator can also view the Amazon CloudFormation template that the SageMaker provided project templates define in the Service Catalog Console. For information about using the Service Catalog console, see [What Is Amazon Service Catalog](#) in the [Amazon Service Catalog User Guide](#).
2. Enable projects for SageMaker studio users who are configured to use the domain execution role. This grants permission for SageMaker Studio users the ability to use the SageMaker provided project templates to create a project from within SageMaker Studio.

To enable users who use any role other than the domain execution role to view and use SageMaker provided project templates, you need to enable **Projects** permissions on the individual user profiles.

The following procedures show how to enable **Projects** permissions when you are onboarding to SageMaker Studio. For more information about onboarding to SageMaker Studio, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).

## To enable Projects permissions for the administrator and domain execution role users

1. Open the [SageMaker console](#).
2. Choose **Amazon SageMaker Studio** at the top left of the page.
3. If you choose **Quick start**, under **Projects**, make sure **Enable SageMaker project templates for this account and Studio users** is enabled.  

4. If you choose **Standard setup**, under **Projects**, make sure both **Enable SageMaker project templates for this account** and **Enable SageMaker project templates for Studio users** are enabled.  


## Create an MLOps Project

Create a SageMaker MLOps project by using SageMaker studio.

### Prerequisites

To create a project in SageMaker Studio, you need:

- An Amazon Web Services SSO or IAM account to sign in to Studio. For information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
- Basic familiarity with the Studio user interface and Jupyter notebooks. For information, see [Amazon SageMaker Studio UI Overview \(p. 69\)](#).

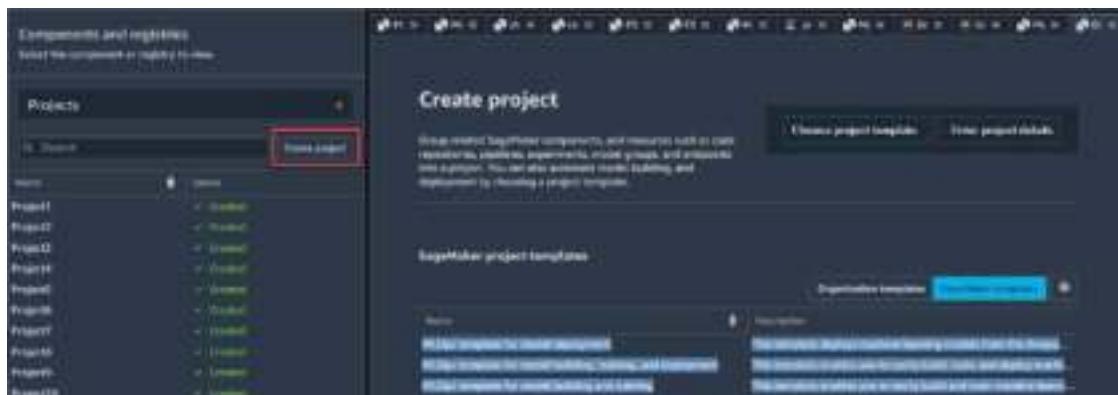
### To create a project in SageMaker Studio

1. Sign in to Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
2. Choose **Components and registries**, and then choose **Projects** in the drop-down list.

The screenshot shows the 'Components and registries' interface in Amazon SageMaker Studio. On the left, there is a vertical toolbar with icons for different components: a folder, a play button, a square, a palette, a wrench, a triangle, a folder, a gear, and a network icon. The 'Projects' icon is highlighted with a red box. The main area has a dark header with the text 'Components and registries' and 'Select the component or registry to view.' Below this is a sub-header 'Projects' with a search bar and a 'Create project' button. A table lists ten projects: Project1 through Project10, all marked as 'Created'. The table has columns for 'Name' and 'Status'.

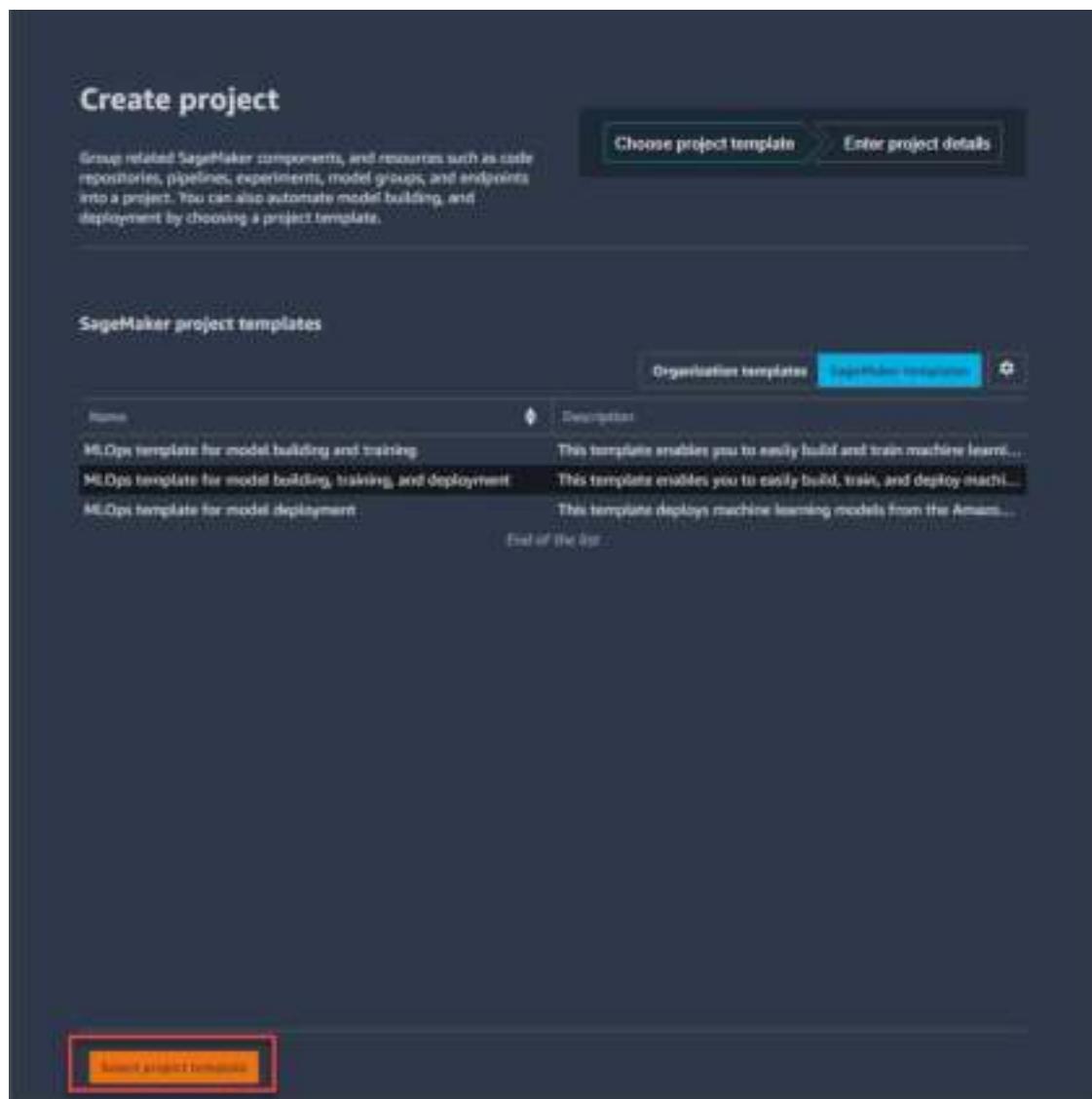
Name	Status
Project1	✓ Created
Project2	✓ Created
Project3	✓ Created
Project4	✓ Created
Project5	✓ Created
Project6	✓ Created
Project7	✓ Created
Project8	✓ Created
Project9	✓ Created
Project10	✓ Created

3. Choose **Create project**.

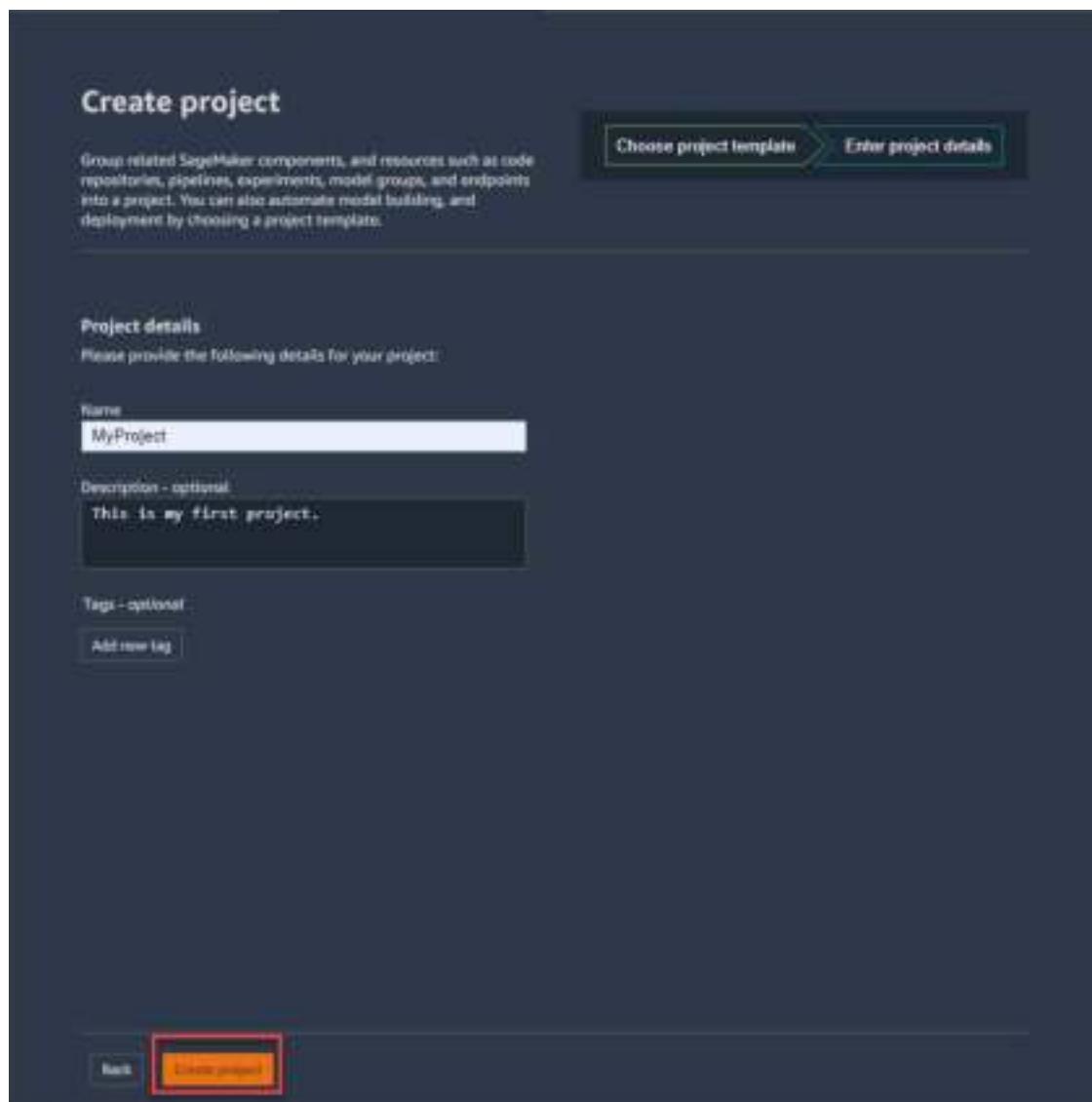


The **Create project** tab appears.

4. For **SageMaker project templates**, choose **Organization templates** to choose a custom template that your organization created, or **SageMaker templates** to choose a SageMaker-provided template. Then choose a template from the list of available templates, and choose **Choose project template**. For more information about project templates, see [MLOps Project Templates \(p. 2240\)](#).



5. For **Project details** enter a name and description for your project. Optionally, add tags, which are key value pairs that you can use to track your projects. When you are finished, choose **Create project**



## MLOps Project Templates

A SageMaker project template automates setting up and implementing MLOps for your projects. A SageMaker project template is an Amazon Service Catalog product that SageMaker makes available to SageMaker Studio users. These Service Catalog products are visible in your Service Catalog console after you enable permissions when you onboard or update SageMaker Studio. For information about enabling permissions to use SageMaker project templates, see [SageMaker Studio Permissions Required to Use Projects \(p. 2236\)](#). Use SageMaker project templates to create a project that is an end-to-end MLOps solution.

If you are an administrator, you can create your own custom project templates. SageMaker Studio users in your organization can use custom project templates that you create to create projects.

### Topics

- [Use SageMaker Provided Project Templates \(p. 2241\)](#)
- [Create Custom Project Templates \(p. 2241\)](#)

## Use SageMaker Provided Project Templates

SageMaker provides project templates that create the infrastructure you need to create an MLOps solution. Currently, SageMaker offers the following project templates.

- **MLOps template for model building and training** - This template enables you to build and train machine learning models and register models to the model registry. Use this template when you want a MLOps solution for building and training models. This template provides the following resources:
  - An Amazon CodeCommit repository that contains sample code that creates SageMaker pipeline in python code and shows how to create and update a SageMaker pipeline. This repository also has a Python Jupyter notebook that you can open and run in SageMaker Studio.
  - A CodePipeline that has source and build steps. The source step points to the CodeCommit repository and the build step gets the code from that repository, creates and/or updates the SageMaker pipeline, starts a pipeline execution, and waits for the pipeline execution to complete.
  - An Amazon S3 bucket to store artifacts, including CodePipeline and CodeBuild artifacts, and any artifacts generated from the SageMaker pipeline runs.

In a collaborative environment with multiple Studio users working on a same project, we recommend creating this project. After data scientists experiment in SageMaker Studio and check their code into CodeCommit, model building and training happens in the common infrastructure so that there is a central, authoritative location to keep track of the ML Models and artifacts that are ready to go to production.

- **MLOps template for model deployment** - This template deploys machine learning models from the Amazon SageMaker model registry to SageMaker hosted endpoints for real-time inference. Use this template when you have trained models that you want to deploy for inference. This template provides the following resources:
  - An Amazon CodeCommit repository that contains sample code that deploys models to endpoints in staging and production environments.
  - A CodePipeline that has source, build, deploy to staging, and deploy to production steps. The source step points to the CodeCommit repository, the build step gets the code from that repository, generates Amazon CloudFormation stacks to deploy. The deploy to staging and deploy to production steps deploy the Amazon CloudFormation stacks to their respective environments. There is a manual approval step between the staging and production build steps, so that a MLOps engineer must approve the model before it is deployed to production.

There is also a programmatic approval step with placeholder tests in the example code in the CodeCommit repository. You can add additional tests to replace the placeholders tests.

- An Amazon S3 bucket to store artifacts, including CodePipeline and CodeBuild artifacts, and any artifacts generated from the SageMaker pipeline runs.

This template recognizes changes in the model registry. When a new model version is registered and approved, it automatically triggers a deployment.

- **MLOps template for model building, training, and deployment** - This template enables you to easily build, train, and deploy machine learning models. Use this template when you want a complete MLOps solution from data preparation to model deployment.

This template is a combination of the previous 2 templates, and contains all of the resources provided in those templates.

## Create Custom Project Templates

If the SageMaker provided templates do not meet your needs (for example, you want to have more complex orchestration in the CodePipeline with multiple stages, custom approval steps, etc.) create your own templates.

We recommend starting by using SageMaker provided templates to understand how to organize your code and resources and build on top of it. To do this, after you enable administrator access to the SageMaker templates, login to the <https://console.amazonaws.cn/servicecatalog/>, choose **Portfolios**, then choose **Imported**. For information about Amazon Service Catalog, see [Overview of Amazon Service Catalog](#) in the *Amazon Service Catalog User Guide*.

Create your own project templates to customize your MLOps project. SageMaker project templates are Amazon Service Catalog provisioned products to provision the resources for your MLOps project.

To create a custom project template, complete the following steps.

1. Create a Portfolio. For information, see [Step 3: Create an Amazon Service Catalog Portfolio](#).
2. Create a Product. A Product is a Amazon CloudFormation template. You can create multiple versions of the product. For information, see [Step 4: Create an Amazon Service Catalog Product](#).

For the Product to work with SageMaker projects, add the following parameters to your Product template.

```
SageMakerProjectName:  
  Type: String  
  Description: Name of the project  
  
SageMakerProjectId:  
  Type: String  
  Description: Service generated Id of the project.
```

3. Add a launch constraint. A launch constraint designates an IAM role that Service Catalog assumes when a user launches a product. For information, see [Step 6: Add a Launch Constraint to Assign an IAM Role](#).
4. Provision the product on the <https://console.amazonaws.cn/servicecatalog/> to test the template. If you are satisfied with your template, continue to the next step to make the template available in SageMaker Studio.
5. Grant access to the Service Catalog Portfolio that you created in step 1 to your SageMaker Studio execution role. Use either the SageMaker Studio domain execution role or a user role that has SageMaker Studio access. For information about adding a role to the Portfolio, see [Step 7: Grant End Users Access to the Portfolio](#).
6. To make your project template available in your **Organization templates** list in SageMaker Studio, create a tag with the following key and value to the Service Catalog Product you created in step 2.
  - **key** - sagemaker:studio-visibility
  - **value** - true

After you complete these steps, SageMaker Studio users in your organization can create a project with the template you created by following the steps in [Create an MLOps Project \(p. 2236\)](#) and choosing **Organization templates** when you choose a template.

## View Project Resources

After you create a project, view the resources associated with the project in SageMaker Studio.

### To create a project in SageMaker Studio

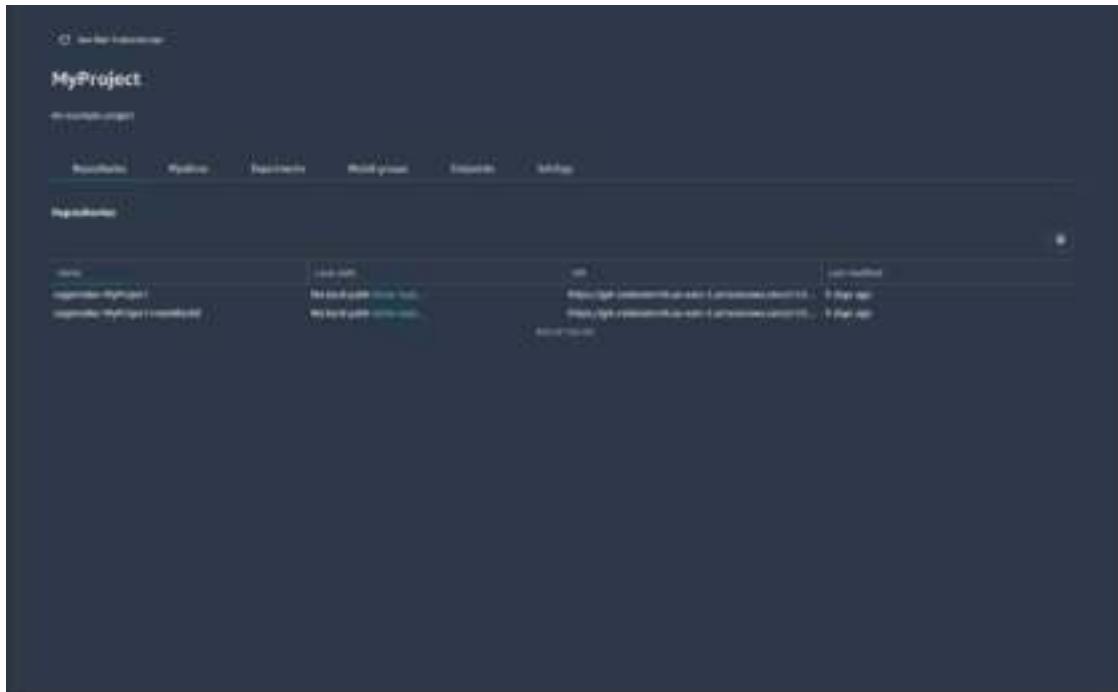
1. Sign in to Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
2. Choose **Components and registries**, and then choose **Projects**.

The screenshot shows the 'Components and registries' section of the Amazon SageMaker console. On the left, there is a vertical sidebar with icons for different resources: a folder, a database, a square, a gear, a key, a network, a file, and a project. The 'Project' icon is highlighted with a red box. The main area has a title 'Components and registries' and a subtitle 'Select the component or registry to view.' Below this is a section titled 'Projects' with a search bar and a 'Create project' button. A table lists ten projects: Project1 through Project10, all marked as 'Created'. The 'Project8' row is also highlighted with a red box.

Name	Status
Project1	✓ Created
Project2	✓ Created
Project3	✓ Created
Project4	✓ Created
Project5	✓ Created
Project6	✓ Created
Project7	✓ Created
Project8	✓ Created
Project9	✓ Created
Project10	✓ Created

3. Double-click the name of the project for which you want to view details.

A tab with the project details appears.



On the product details tab, you can view the following entities associated with the project.

- **Repositories** - code repositories associated with this project. If you use a SageMaker provided template when you create your project, it creates Amazon CodeCommit pipelines. For more information about CodeCommit, see [What is Amazon CodeCommit](#)
- **Pipelines** - SageMaker ML pipelines that define steps to prepare data, train, and deploy models. For information about SageMaker ML pipelines, see [Create and Manage SageMaker Pipelines \(p. 2201\)](#).
- **Experiments** - one or more SageMaker AutoPilot experiments associated with the project. For information about AutoPilot, see [Automate model development with Amazon SageMaker Autopilot \(p. 142\)](#).
- **Model groups**- groups of model versions that were created by pipeline executions in the project. For information about model groups, see [Create a Model Group \(p. 2003\)](#).
- **Endpoints** - SageMaker endpoints that host deployed models for real-time inference. When a model version is approved, it is deployed to an endpoint.
- **Settings** - Settings for the project. This includes the name and description of the project, information about the project template and SourceModelPackageName, and metadata about the project.

## SageMaker MLOps Project Walkthrough

This walkthrough demonstrates how to use MLOps projects to create a CI/CD system to build, train, and deploy models.

### Prerequisites

To complete this walkthrough, you need:

- An Amazon Web Services SSO or IAM account to sign in to Studio. For information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
- Permission to use SageMaker provided project templates. For information, see [SageMaker Studio Permissions Required to Use Projects \(p. 2236\)](#).

- Basic familiarity with the Studio user interface. For information, see [Amazon SageMaker Studio UI Overview \(p. 69\)](#).

#### Topics

- [Step 1: Create the Project \(p. 2245\)](#)
- [Step 2: Clone the Code Repository \(p. 2248\)](#)
- [Step 3: Make a Change in the Code \(p. 2250\)](#)
- [Step 4: Approve the Model \(p. 2253\)](#)
- [\(Optional\) Step 5: Deploy the Model Version to Production \(p. 2255\)](#)
- [Step 6: Cleanup Resources \(p. 2256\)](#)

## Step 1: Create the Project

In this step, you create a SageMaker MLOps project by using a SageMaker provided project template to build, train, and deploy models.

### To create the SageMaker MLOps project

1. Sign in to Studio. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#).
2. Choose **Components and registries**, and then choose **Projects** in the drop-down list.

The screenshot shows the 'Components and registries' section of the SageMaker console. On the left is a vertical toolbar with icons for different components: folder, ML model, Lambda function, S3 bucket, Kinesis stream, and a key icon. Below the toolbar, the title 'Projects' is displayed above a search bar and a 'Create project' button. A table lists ten projects: Project1 through Project10, all in a 'Created' status. The 'key' icon in the toolbar is highlighted with a red box.

Name	Status
Project1	✓ Created
Project2	✓ Created
Project3	✓ Created
Project4	✓ Created
Project5	✓ Created
Project6	✓ Created
Project7	✓ Created
Project8	✓ Created
Project9	✓ Created
Project10	✓ Created

3. Choose **Create project**.

The screenshot shows the 'Create project' tab. The left sidebar displays a list of existing projects: Project1, Project2, Project3, Project4, Project5, Project6, Project7, Project8, Project9, and Project10. The 'Create project' tab itself has two tabs: 'Create project template' (selected) and 'Enter project details'. Below these tabs, there is descriptive text about creating a project and a list of 'SageMaker project templates'.

**Create project**

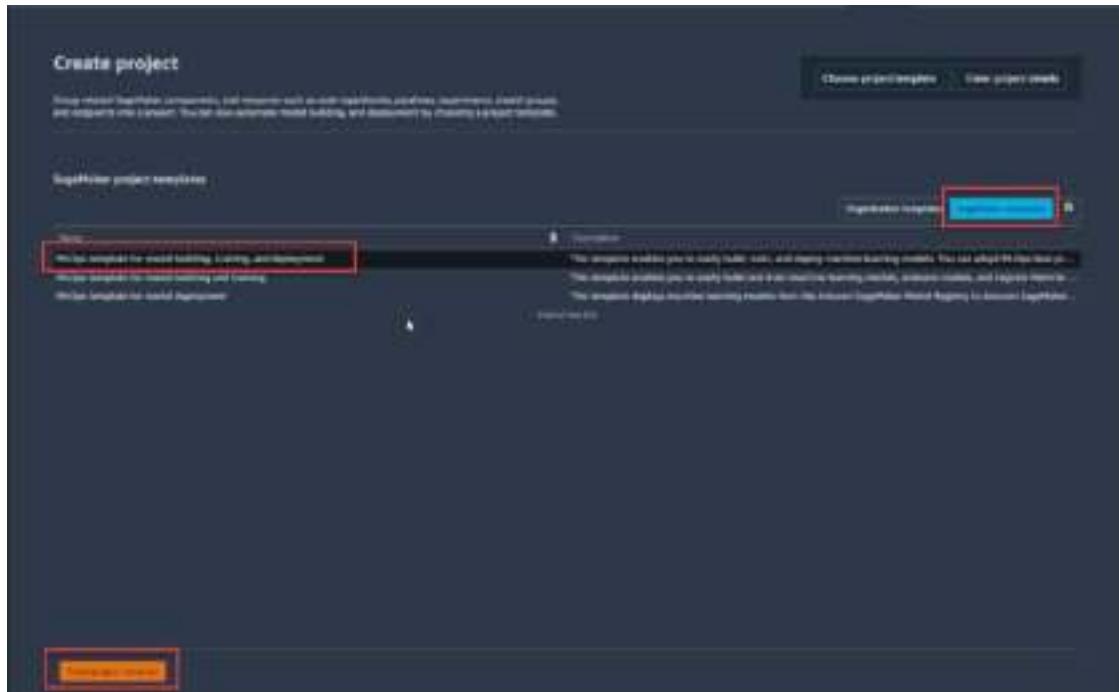
Create your first SageMaker components and resources such as code, training jobs, inference endpoints, model groups, and pipelines. With a project template, you can quickly build, train, and deploy by choosing a project template.

**SageMaker project templates**

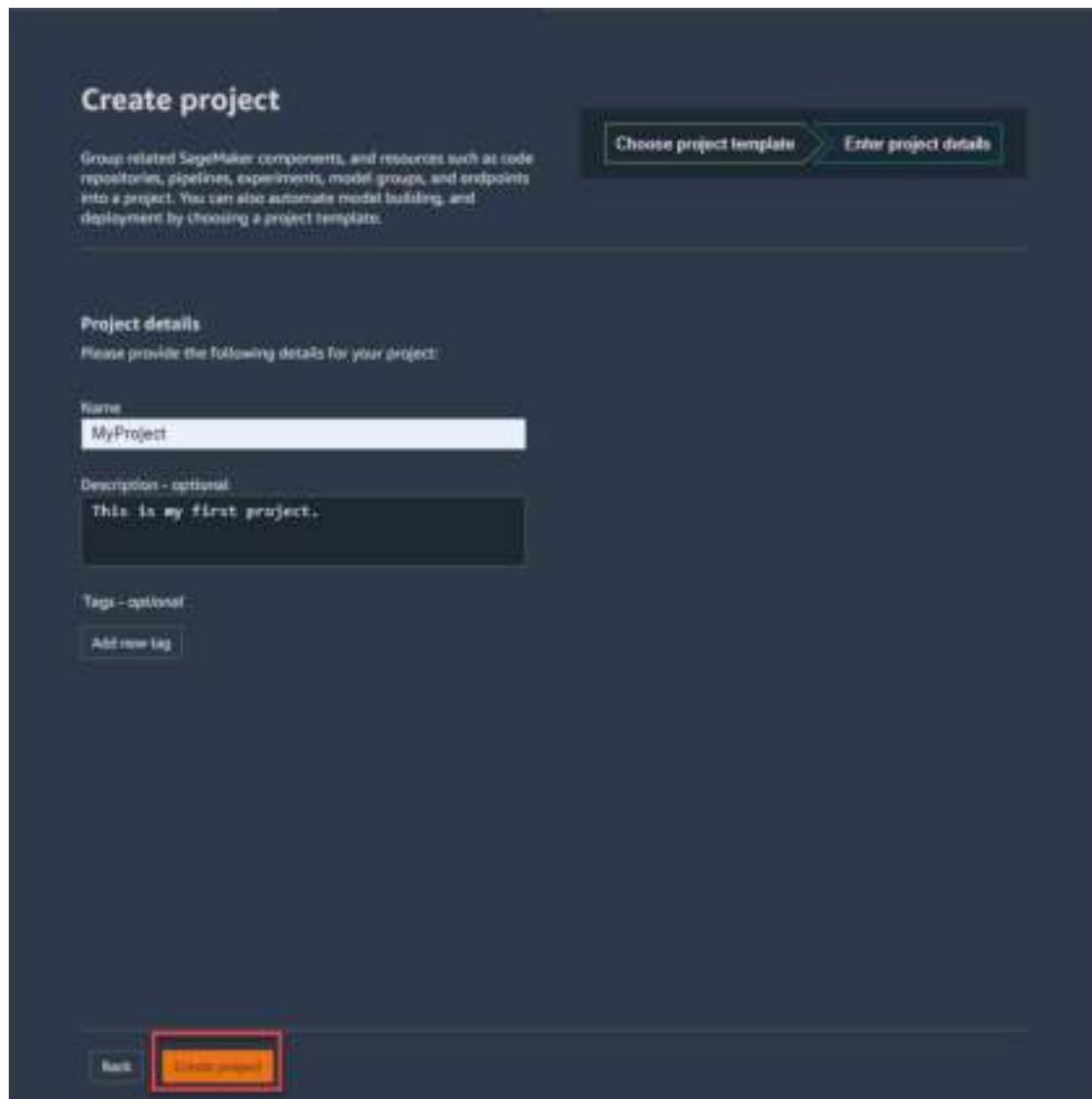
Name	Description
ML Project template for model training	ML Project template for model training
ML Project template for endpoint configuration and deployment	ML Project template for endpoint configuration and deployment
ML Project template for model hosting and serving	ML Project template for model hosting and serving
ML Project template for model training and deployment	ML Project template for model training and deployment

The **Create project** tab appears.

4. For **SageMaker project templates**, choose **Organization templates**, then choose **MLOps template for model building, training, and deployment**.



5. For **Project details** enter a name and description for your project. **Create project**



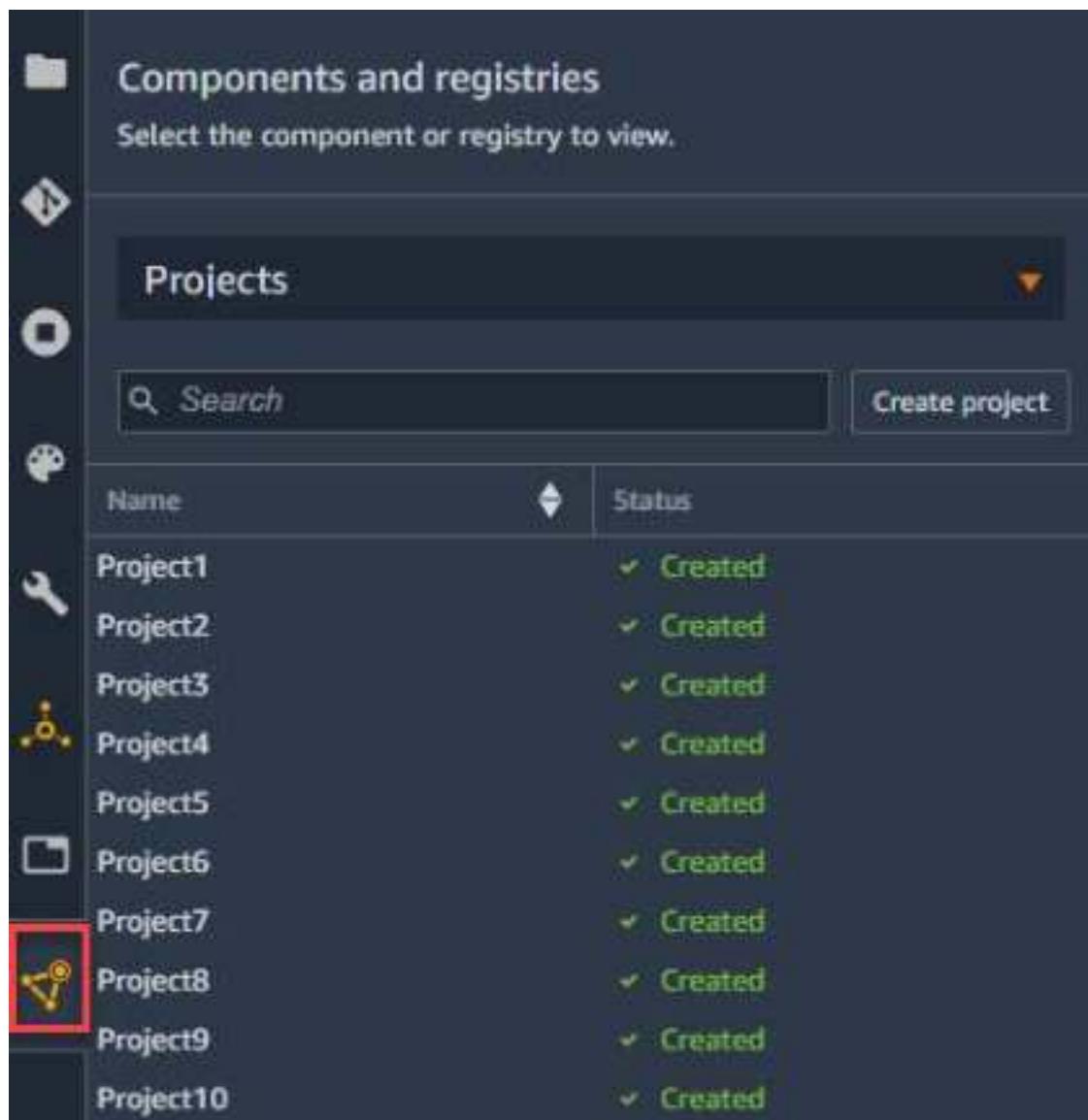
When the project appears in the **Projects** list with a **Status of Created**, move on to the next step.

## Step 2: Clone the Code Repository

After you create the project, two CodeCommit repositories are created in the project. One of the repositories contains code to build and train a model, and one contains code to deploy the model. In this step, you clone the repository to the local SageMaker that contains the code to build and train the model to the local SageMaker Studio environment so that you can work with the code.

### To clone the code repository

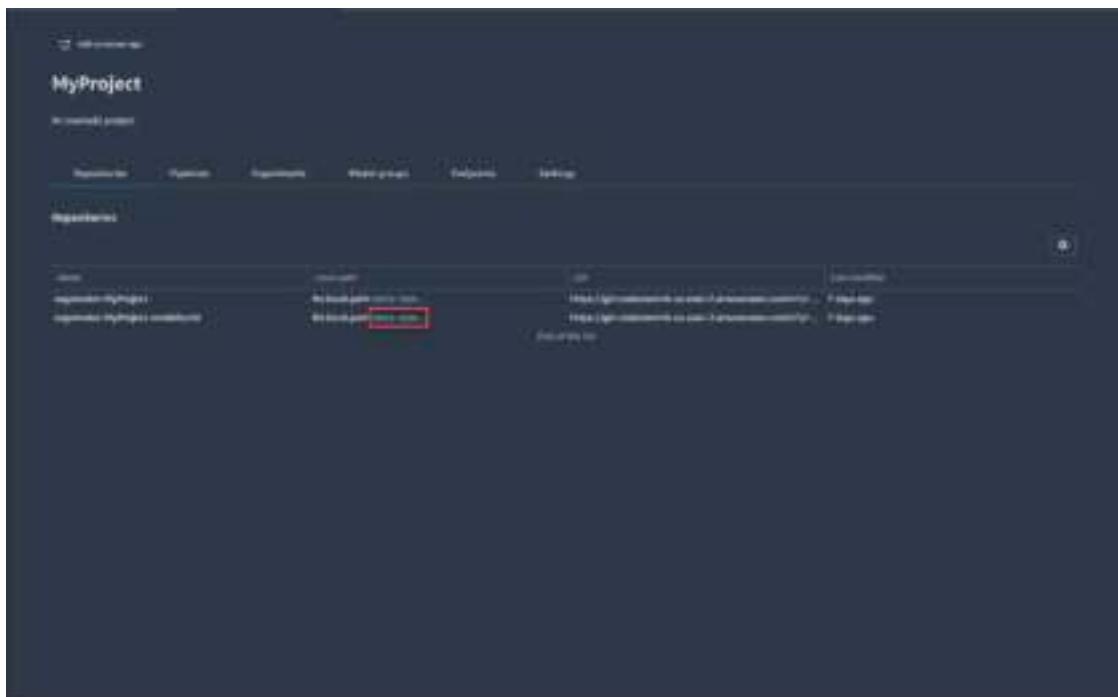
1. Choose **Components and registries**, and then choose **Projects** in the drop-down list.



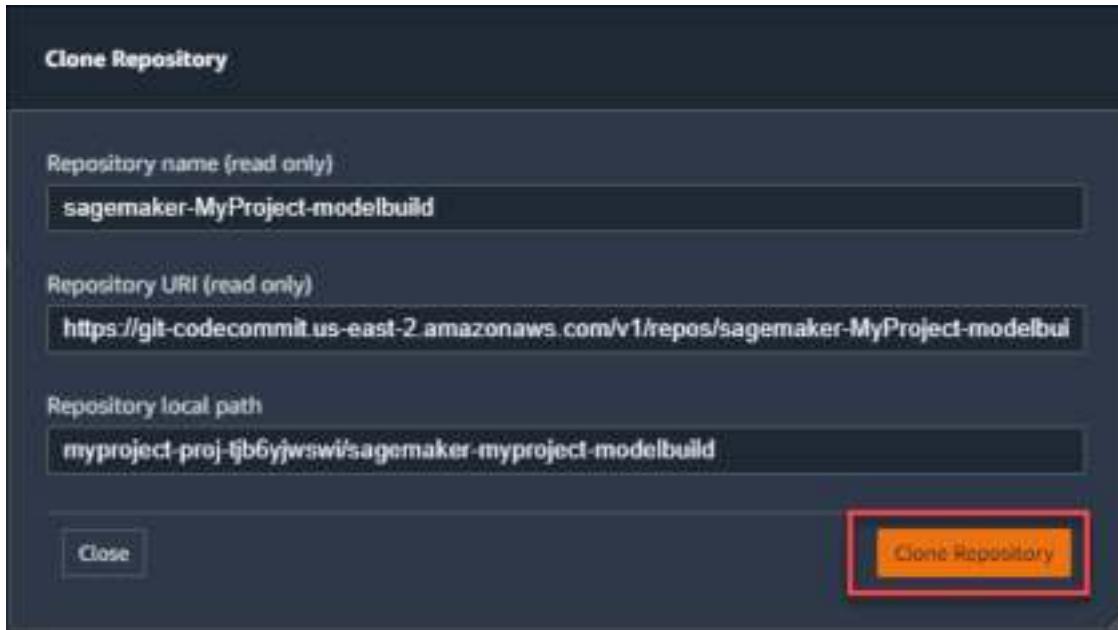
The screenshot shows the 'Components and registries' section of the Amazon SageMaker console. On the left, there is a vertical sidebar with icons for different components: a folder, a database, a square, a gear, a key, a network, a folder, and a magnifying glass. The magnifying glass icon is highlighted with a red box. The main area has a dark header with the title 'Components and registries' and the sub-instruction 'Select the component or registry to view.' Below this is a sub-header 'Projects' with a dropdown arrow. There is a search bar labeled 'Search' and a 'Create project' button. A table lists ten projects, each with a small icon and a status indicator:

Name	Status
Project1	✓ Created
Project2	✓ Created
Project3	✓ Created
Project4	✓ Created
Project5	✓ Created
Project6	✓ Created
Project7	✓ Created
Project8	✓ Created
Project9	✓ Created
Project10	✓ Created

2. Find the name of the project you created in the previous step and double-click on it to open the project tab for your project.
3. In the project tab, choose **Repositories**, and in the **Local path** column for the repository that ends with **modelbuild**, choose **clone repo...**



4. In the dialog box that appears, accept the defaults and choose **Clone repository**.



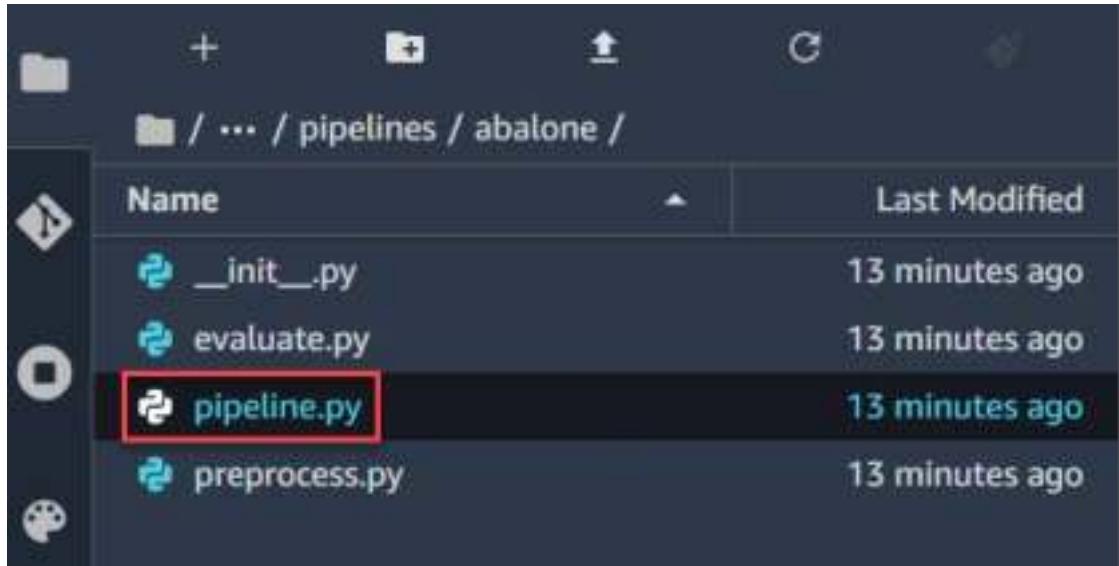
When clone of the repository is complete, the local path appears in the **Local path** column. Click on the path to open the local folder that contains the repository code in SageMaker Studio.

### Step 3: Make a Change in the Code

Now make a change to the pipeline code that builds the model and check in the change to trigger a new pipeline run. The pipeline run registers a new model version.

### To make a code change

1. In SageMaker Studio, choose the file browser icon (  ), and navigate to the pipelines/abalone folder. Double-click pipeline.py to open the code file.



2. In the pipeline.py file, find the line that sets the training instance type.

```
training_instance_type = ParameterString(  
    name="TrainingInstanceType", default_value="ml.m5.xlarge"
```

Change ml.m5.xlarge to ml.m5.large, then type Ctrl+S to save the change.

3. Choose the Git icon (  ). Stage, commit, and push the change in pipeline.py. For information about using Git in SageMaker Studio, see [Clone a Git Repository in SageMaker Studio \(p. 114\)](#).

The screenshot shows a Git commit interface with the following details:

- Current Repository:** sagemaker-myproject-p-kjwrgqlr05qn-modelbuild
- Current Branch:** master
- Changes Tab:** The "Changes" tab is selected.
- Staged:** 1 file: pipeline.py (M)
- Changed:** 0 files (0)
- Untracked:** 4 files:
  - .sagemaker-code-config (U)
  - Untitled-checkpoint.ipynb (U)
  - pipeline-checkpoint.py (U)

A message box at the bottom left says "Changed instance type".

The bottom right corner shows a red box around the "Commit" button, which is highlighted in blue.

At the very bottom, there are icons for terminal, file count (0), and Python, along with the text "Git: refreshing..." and the number 2252.

After pushing your code change, the MLOps system triggers a run of the pipeline that creates a new model version. In the next step, you approve the new model version to deploy it to production.

## Step 4: Approve the Model

Now you approve the new model version that was created in the previous step to trigger a deployment of the model version to a SageMaker endpoint.

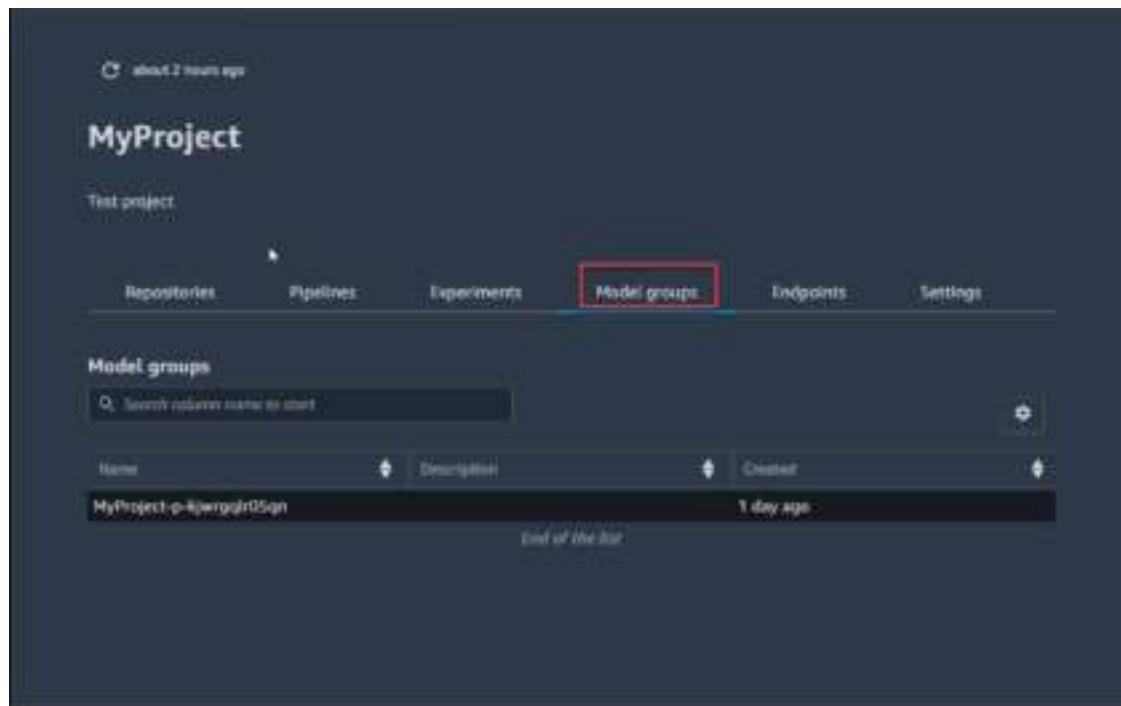
### To approve the model version

1. Choose **Components and registries**, and then choose **Projects** in the drop-down list.

The screenshot shows the 'Components and registries' interface with a sidebar containing icons for components like Lambda, Step Functions, and ML Models. The main area is titled 'Components and registries' with the sub-section 'Projects'. A search bar and a 'Create project' button are at the top of the list. The table below lists ten projects, each with a status of 'Created'.

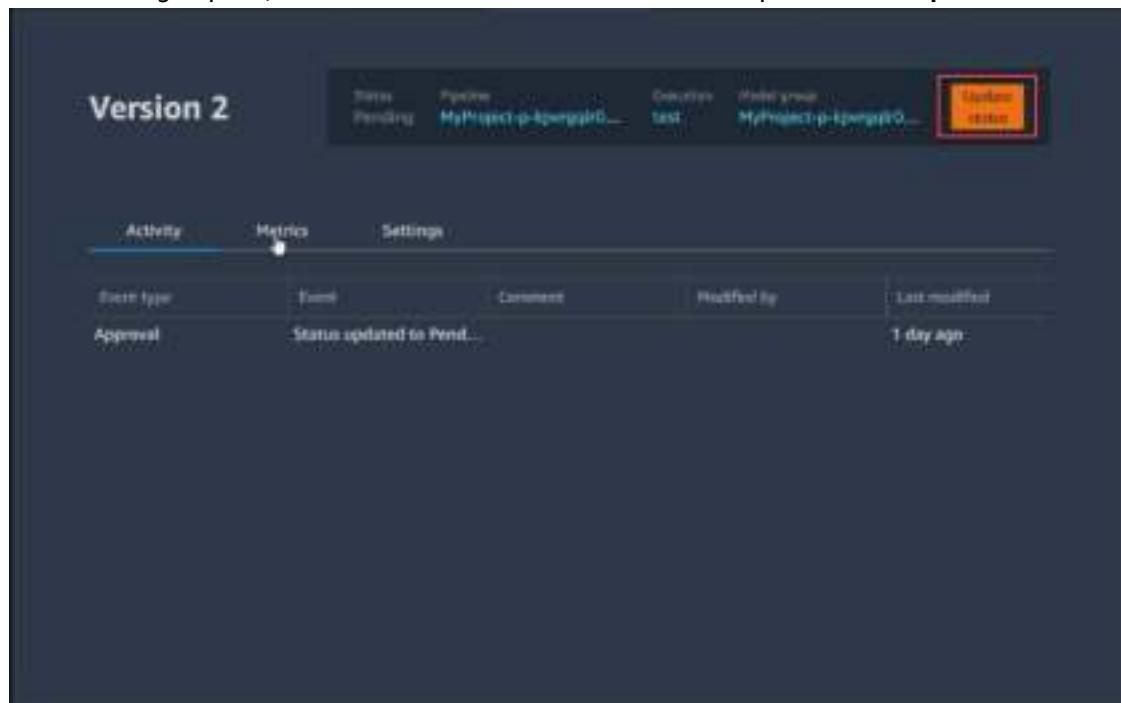
Name	Status
Project1	✓ Created
Project2	✓ Created
Project3	✓ Created
Project4	✓ Created
Project5	✓ Created
Project6	✓ Created
Project7	✓ Created
Project8	✓ Created
Project9	✓ Created
Project10	✓ Created

2. Find the name of the project you created in the first step and double-click on it to open the project tab for your project.
3. In the project tab, choose **Model groups**, then double-click the name of the model group that appears.

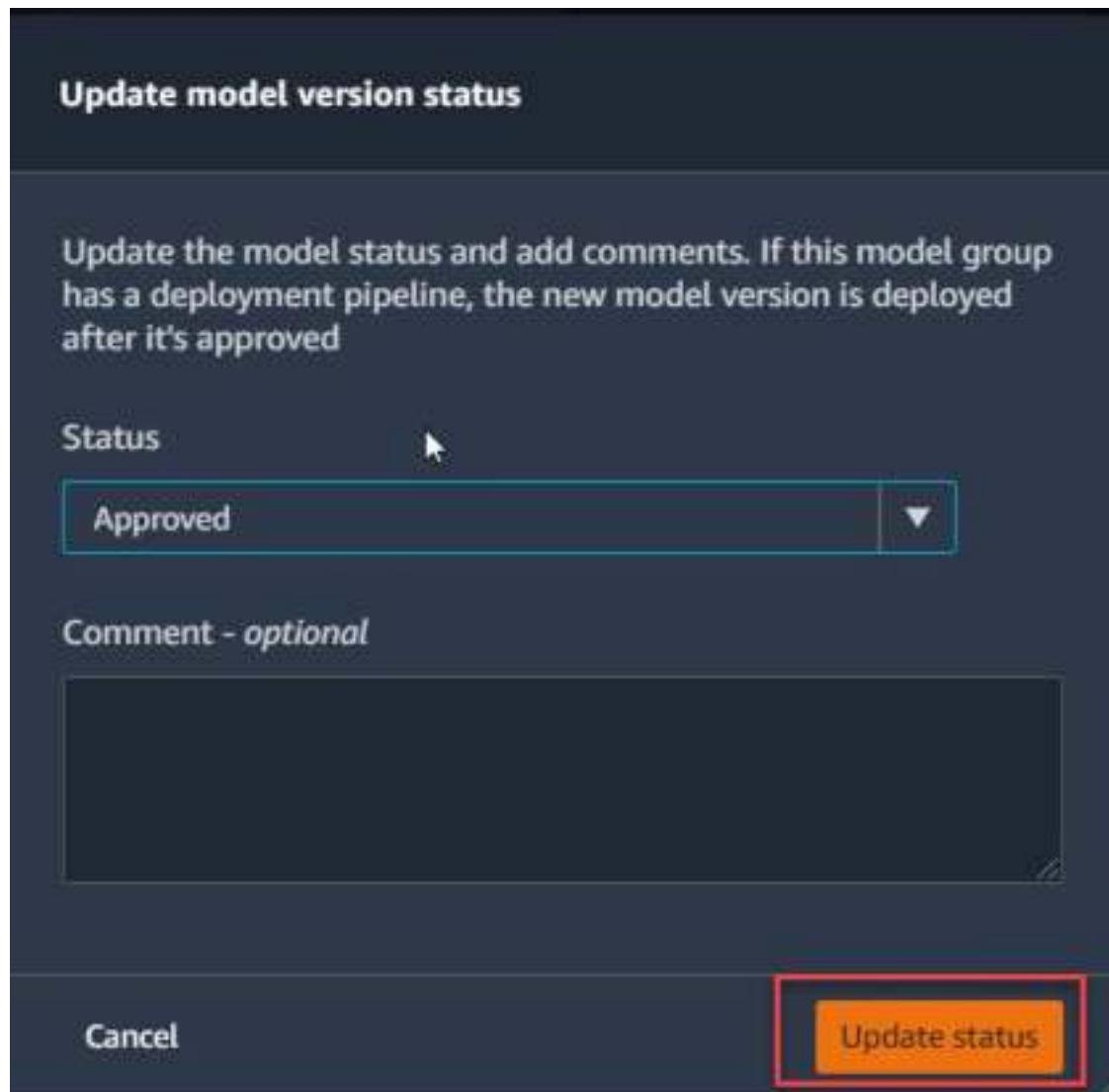


The model group tab appears.

4. In the model group tab, double-click **Version 2**. The **Version 2** tab opens. Choose **Update status**



5. In the model **Update model version status** dialog box, in the **Status** drop-down, choose **Approve**, then choose **Update status**.



Approving the model version causes the MLOps system to deploy the model to staging. To view the endpoint, choose the **Endpoints** tab on the project tab.

## (Optional) Step 5: Deploy the Model Version to Production

Now you can deploy the model version to the production environment.

**Note**

To complete this step, you need to be an administrator in your SageMaker Studio domain. If you are not an administrator, skip this step.

### To deploy the model version to the production environment

1. Log in to the CodePipeline console at <https://console.amazonaws.cn/codepipeline/>
2. Choose **Pipelines**, then choose the pipeline with the name **sagemaker-*projectname-projectid-modeldeploy***, where *projectname* is the name of your project, and *projectid* is the ID of your project.
3. In the **DeployStaging** stage, choose **Review**.

4. In the **Review** dialog box, choose **Approve**.

Approving the **DeployStaging** stage causes the MLOps system to deploy the model to production. To view the endpoint, choose the **Endpoints** tab on the project tab in SageMaker Studio.

## Step 6: Cleanup Resources

To stop incurring charges, clean up the resources that were created in this walkthrough. To do this, complete the following steps.

**Note**

To delete the Amazon CloudFormation stack and the Amazon S3 bucket, you need to be an administrator in SageMaker Studio. If you are not an administrator, ask your administrator to complete those steps.

1. From the SageMaker Studio menu, choose **File**, choose **New**, and then choose **Notebook**.
2. In the **Select Kernel** dialog box, choose **Python 3 (Data Science)**, then choose **Select**.
3. In the notebook, enter the following code in a cell, and then run the cell. Replace **MyProject** with the name of your project.

```
import boto3

sm_client=boto3.client("sagemaker")
sm_client.delete_project(ProjectName="MyProject")
```

This deletes the Service Catalog provisioned product that the project created. This includes the CodeCommit, CodePipeline, and CodeBuild resources that were created for the project.

4. Delete the Amazon CloudFormation stacks that the project created. There are 2 stacks, one for staging and one for production. The names of the stacks are **sagemaker-projectname-project-id-deploy-staging** and **sagemaker-projectname-project-id-deploy-prod**, where **projectname** is the name of your project, and **project-id** is the ID of your project.

For information about how to delete a Amazon CloudFormation stack, see [Deleting a stack on the Amazon CloudFormation console](#) in the *Amazon CloudFormation User Guide*.

5. Delete the Amazon S3 bucket that the project created. The name of the bucket is **sagemaker-project-project-id**, where **project-id** is the ID of your project.

## Amazon SageMaker ML Lineage Tracking

Amazon SageMaker ML Lineage Tracking creates and stores information about the steps of a machine learning (ML) workflow from data preparation to model deployment. With the tracking information you can reproduce the workflow steps, track model and dataset lineage, and establish model governance and audit standards.

With SageMaker Lineage Tracking data scientists and model builders can do the following:

- Keep a running history of model discovery experiments.
- Establish model governance by tracking model lineage artifacts for auditing and compliance verification.
- Clone and rerun workflows to experiment with what-if scenarios while developing models.
- Share a workflow that colleagues can reproduce and enhance (for example, while collaborating on solving a business problem).

- Clone and rerun workflows with additional debugging or logging routines, or new input variations for troubleshooting issues in production models.

#### Topics

- [Tracking Entities \(p. 2257\)](#)
- [Amazon SageMaker Created Tracking Entities \(p. 2258\)](#)
- [Manually Create Tracking Entities \(p. 2260\)](#)

## Tracking Entities

Tracking entities maintain a representation of all the elements of your end-to-end machine learning workflow. You can use this representation to establish model governance, reproduce your workflow, and maintain a record of your work history.

Amazon SageMaker automatically creates tracking entities for trial components and their associated trials and experiments when you create SageMaker jobs such as processing jobs, training jobs, and batch transform jobs. For more information, see [Manage Machine Learning with Amazon SageMaker Experiments \(p. 1553\)](#).

SageMaker also automatically creates tracking entities for the other steps in a workflow that enable you to track the workflow from end to end. For more information, see [Amazon SageMaker Created Tracking Entities \(p. 2258\)](#).

You can create additional entities to supplement those created by SageMaker. For more information, see [Manually Create Tracking Entities \(p. 2260\)](#).

SageMaker reuses any existing entities rather than create new ones. For example, there can be only one artifact with a unique `SourceUri`.

The following tracking entities are defined:

### Experiment entities

- **Trial component** – A stage of a machine learning trial. Includes processing jobs, training jobs, and batch transform jobs.
- **Trial** – A combination of trial components that generally produces a model.
- **Experiment** – A grouping of trials generally focused on solving a specific use case.

### Lineage entities

- **Context** – Provides a logical grouping of other tracking or experiment entities. Conceptually, experiments and trials are contexts. Some examples are an endpoint and a model package.
- **Action** – Represents an action or activity. Generally, an action involves at least one input artifact or output artifact. Some examples are a workflow step and a model deployment.
- **Artifact** – Represents a URI addressable object or data. An artifact is generally either an input or an output to a trial component or Action. Some examples include a dataset (Amazon S3 bucket URI), an image (Amazon ECR registry path), or an action (ARN).
- **Association** – Links other tracking or experiment entities. For example, an association between the location of training data and a training job.

An association has an optional `AssociationType` property. The following values are available along with the suggested use for each type. SageMaker places no restrictions on their use:

- **ContributedTo** – The source contributed to the destination or had a part in enabling the destination. For example, the training data contributed to the training job.

- **AssociatedWith** – The source is connected to the destination. For example, an approval workflow is associated with a model deployment.
- **DerivedFrom** - The destination is a modification of the source. For example, a digest output of a channel input for a processing job is derived from the original inputs.
- **Produced** – The source generated the destination. For example, a training job produced a model artifact.

### Common properties

- **Type property**

The action, artifact, and context entities have a `type` property, `ActionType`, `ArtifactType`, and `ContextType`, respectively. This property is a custom string which can associate meaningful information with the entity and be used as a filter in the List APIs.

- **Source property**

The action, artifact, and context entities have a `Source` property. This property provides the underlying URI that the entity represents. Some examples are:

- An `UpdateEndpoint` action where the source is the `EndpointArn`.
- An image artifact for a processing job where the source is the `ImageUri`.
- An Endpoint context where the source is the `EndpointArn`.

- **Metadata property**

The action and artifact entities have an optional `Metadata` property which can provide the following information:

- `ProjectId` – For example, the ID of the SageMaker MLOps project a model belongs to.
- `GeneratedBy` – For example, the SageMaker pipeline execution that registered a model package version.
- `Repository` – For example, the repository that contains an algorithm.
- `CommitId` – For example, the commit ID of an algorithm version.

## Amazon SageMaker Created Tracking Entities

Amazon SageMaker automatically creates tracking entities for SageMaker jobs, models, model packages, and endpoints if the data is available. There is no limit to the number of lineage entities automatically created by SageMaker.

For information on how you can manually create tracking entities, see [Manually Create Tracking Entities \(p. 2260\)](#).

### Topics

- [Tracking Entities for SageMaker Jobs \(p. 2258\)](#)
- [Tracking Entities for Model Packages \(p. 2259\)](#)
- [Tracking Entities for Endpoints \(p. 2259\)](#)

## Tracking Entities for SageMaker Jobs

A trial component is created for and associated with each SageMaker job. Artifacts are created to track the job metadata. Associations are created between each artifact and the job.

Artifacts are created for the following job properties and associated with the Amazon Resource Name (ARN) of the SageMaker job. The artifact `SourceUri` is listed in parentheses.

### Training Job

- The image that contains the training algorithm (`TrainingImage`)
- The data source of each input channel (`S3Uri`)
- The location for the model (`S3OutputPath`)
- The location for the managed spot checkpoint data (`s3uri`)

### Processing Job

- The container to be run by the processing job (`ImageUri`)
- The data location for each processing input and processing output (`s3uri`)

### Transform Job

- The input data source to be transformed (`S3Uri`)
- The results of the transform (`S3OutputPath`)

#### Note

Amazon Simple Storage Service (Amazon S3) artifacts are tracked based on the S3 URI values provided to the create API, for example [CreateTrainingJob](#), and not on the S3 key and hash/etag values from each file.

## Tracking Entities for Model Packages

The following entities are created:

### Model Packages

- A context for each model package group
- An artifact for each model package
- An association between each model package artifact and the context for each model package group the package belongs to
- An action for the creation of a model package version
- An association between the model package artifact and the creation action
- An association between the model package artifact and each model package group context the package belongs to
- Inference containers
  - An artifact for the image used in each container defined in the model package
  - An artifact for the model used in each container
  - An association between each artifact and the model package artifact
- Algorithms
  - An artifact for each algorithm defined in the model package
  - An artifact for the model created by each algorithm
  - An association between each artifact and the model package artifact

## Tracking Entities for Endpoints

The following entities are created:

## Endpoints

- A context for each endpoint
- An action for the model deployment that created each endpoint
- An artifact for each model deployed to the endpoint
- An artifact for the image used in the model
- An artifact for the model package for the model
- An artifact for each image deployed to the endpoint
- An association between each artifact and the model deployment action

## Manually Create Tracking Entities

You can manually create tracking entities for any property. For information on the tracking entities that Amazon SageMaker automatically creates, see [Amazon SageMaker Created Tracking Entities \(p. 2258\)](#).

You can add tags to all entities except associations. Tags are arbitrary key-value pairs that provide custom information. You can filter or sort a list or search query by tags. For more information, see [Tagging Amazon resources](#) in the *Amazon General Reference*.

For a sample notebook that demonstrates how to create lineage entities, see the [Amazon SageMaker Lineage](#) notebook in the [Amazon SageMaker example GitHub repository](#).

### Topics

- [Manually Create Entities \(p. 2260\)](#)
- [Manually Track a Workflow \(p. 2262\)](#)
- [Limits \(p. 2263\)](#)

## Manually Create Entities

The following procedure shows you how to create and associate artifacts between a SageMaker training job and endpoint. You perform the following steps:

- Create input artifacts for the source code, training data, and testing data locations.
- Create an output artifact for the generated model location.
- Create a trial component as a training job.
- Associate the input artifact and output artifacts with the training job.
- Train the model and create an endpoint.
- Create a context for the endpoint.
- Associate the training job and the endpoint context.

### To create tracking entities and associations

1. Import the tracking entities.

```
import sys
!{sys.executable} -m pip install -q sagemaker

from sagemaker import get_execution_role
from sagemaker.session import Session
from sagemaker.lineage import context, artifact, association, action
```

```
import boto3
boto_session = boto3.Session(region_name=region)
sagemaker_client = boto_session.client("sagemaker")
```

2. Create the input and output artifacts.

```
code_location_arn = artifact.Artifact.create(
    artifact_name='source-code-location',
    source_uri='s3://...',
    artifact_type='code-location'
).artifact_arn

# Similar constructs for train_data_location_arn and test_data_location_arn

model_location_arn = artifact.Artifact.create(
    artifact_name='model-location',
    source_uri='s3://...',
    artifact_type='model-location'
).artifact_arn
```

3. Train the model and get the trial\_component\_arn that represents the training job.
4. Associate the input artifacts and output artifacts with the training job (trial component).

```
input_artifacts = [code_location_arn, train_data_location_arn, test_data_location_arn]
for artifact_arn in input_artifacts:
    try:
        association.Association.create(
            source_arn=artifact_arn,
            destination_arn=trial_component_arn,
            association_type='ContributedTo'
        )
    except:
        logging.info('association between {} and {} already exists', artifact_arn,
                    trial_component_arn)

output_artifacts = [model_location_arn]
for artifact_arn in output_artifacts:
    try:
        association.Association.create(
            source_arn=trial_component_arn,
            destination_arn=artifact_arn,
            association_type='Produced'
        )
    except:
        logging.info('association between {} and {} already exists', artifact_arn,
                    trial_component_arn)
```

5. Create the inference endpoint.

```
predictor = mnist_estimator.deploy(initial_instance_count=1,
                                    instance_type='ml.m4.xlarge')
```

6. Create the endpoint context.

```
from sagemaker.lineage import context

endpoint = sagemaker_client.describe_endpoint(EndpointName=predictor.endpoint_name)
endpoint_arn = endpoint['EndpointArn']

endpoint_context_arn = context.Context.create(
    context_name=predictor.endpoint_name,
    context_type='Endpoint',
```

```
    source_uri=endpoint_arn
).context_arn
```

7. Associate the training job (trial component) and endpoint context.

```
association.Association.create(
    source_arn=trial_component_arn,
    destination_arn=endpoint_context_arn
)
```

## Manually Track a Workflow

You can manually track the workflow created in the previous section.

Given the endpoint Amazon Resource Name (ARN) from the previous example, the following procedure shows you how to track the workflow back to the datasets used to train the model that was deployed to the endpoint. You perform the following steps:

- Given the endpoint ARN, get the endpoint context.
- Get the trial component from the association between the trial component and the endpoint context.
- Get the training data location artifact from the association between the trial component and the endpoint context.
- Get the training data location from the training data location artifact.

### To track a workflow from endpoint to training data source

1. Import the tracking entities.

```
import sys
!{sys.executable} -m pip install -q sagemaker

from sagemaker import get_execution_role
from sagemaker.session import Session
from sagemaker.lineage import context, artifact, association, action

import boto3
boto_session = boto3.Session(region_name=region)
sagemaker_client = boto_session.client("sagemaker")
```

2. Get the endpoint context from the endpoint ARN.

```
endpoint_context_arn = sagemaker_client.list_contexts(
    SourceUri=endpoint_arn)[ 'ContextSummaries' ][0][ 'ContextArn' ]
```

3. Get the trial component from the association between the trial component and the endpoint context.

```
trial_component_arn = sagemaker_client.list_associations(
    DestinationArn=endpoint_context_arn)[ 'AssociationSummaries' ][0][ 'SourceArn' ]
```

4. Get the training data location artifact from the association between the trial component and the endpoint context.

```
train_data_location_artifact_arn = sagemaker_client.list_associations(
    DestinationArn=trial_component_arn, SourceType='Model')[ 'AssociationSummaries' ][0]
[ 'SourceArn' ]
```

5. Get the training data location from the training data location artifact.

```
train_data_location = sagemaker_client.describe_artifact(  
    ArtifactArn=train_data_location_artifact_arn)['Source']['SourceUri']  
print(train_data_location)
```

Response:

```
s3://sagemaker-sample-data-us-east-2/mxnet/mnist/train
```

## Limits

An association can be created between any entities, experiment and lineage, except the following:

- An association can't be created between two experiment entities. Experiment entities consist of experiments, trials, and trial components.
- An association can't be created with another association.

An error occurs if you try to create an entity that already exists.

### Maximum number of manually created lineage entities

- Actions: 3000
- Artifacts: 6000
- Associations: 6000
- Contexts: 500

There is no limit to the number of lineage entities automatically created by SageMaker.

## Kubernetes Orchestration

You can orchestrate your SageMaker training and inference jobs with SageMaker Operators for Kubernetes and SageMaker Components for Kubeflow Pipelines. SageMaker Operators for Kubernetes make it easier for developers and data scientists using Kubernetes to train, tune, and deploy machine learning (ML) models in SageMaker. SageMaker Components for Kubeflow Pipelines allow you to move your data processing and training jobs from the Kubernetes cluster to SageMaker's machine learning-optimized managed service.

### Contents

- [SageMaker Operators for Kubernetes \(p. 2263\)](#)
- [SageMaker Components for Kubeflow Pipelines \(p. 2294\)](#)

## SageMaker Operators for Kubernetes

SageMaker Operators for Kubernetes make it easier for developers and data scientists using Kubernetes to train, tune, and deploy machine learning (ML) models in SageMaker. You can install these SageMaker Operators on your Kubernetes cluster in Amazon Elastic Kubernetes Service (Amazon EKS) to create SageMaker jobs natively using the Kubernetes API and command-line Kubernetes tools such as kubectl.

This guide shows you how to set up the operators. The guide also explains how to use the operators to run model training, hyperparameter tuning, and inference (real-time and batch).

There is no additional charge to use these operators. You do incur charges for any SageMaker resources that you use through these operators. The procedures and guidelines here assume you are familiar with Kubernetes and its basic commands.

## Contents

- [What is an operator? \(p. 2264\)](#)
- [IAM role-based setup and operator deployment \(p. 2265\)](#)
- [Delete operators \(p. 2274\)](#)
- [Troubleshooting \(p. 2276\)](#)
- [Images and SMlogs in each Region \(p. 2276\)](#)
- [Using SageMaker Jobs \(p. 2277\)](#)

## What is an operator?

Kubernetes is built on top of what is called the *controller pattern*. This pattern allows applications and tools to listen to a central state manager (ETCD) and act when something happens. Examples of such applications include `cloud-controller-manager` and `controller-manager`. The controller pattern allows you to create decoupled experiences and not have to worry about how other components are integrated. To add new capabilities to Kubernetes, developers can extend the Kubernetes API by creating a custom resource that contains their application-specific or domain-specific logic and components. Operators in Kubernetes allow users to natively invoke these custom resources and automate associated workflows.

## Prerequisites

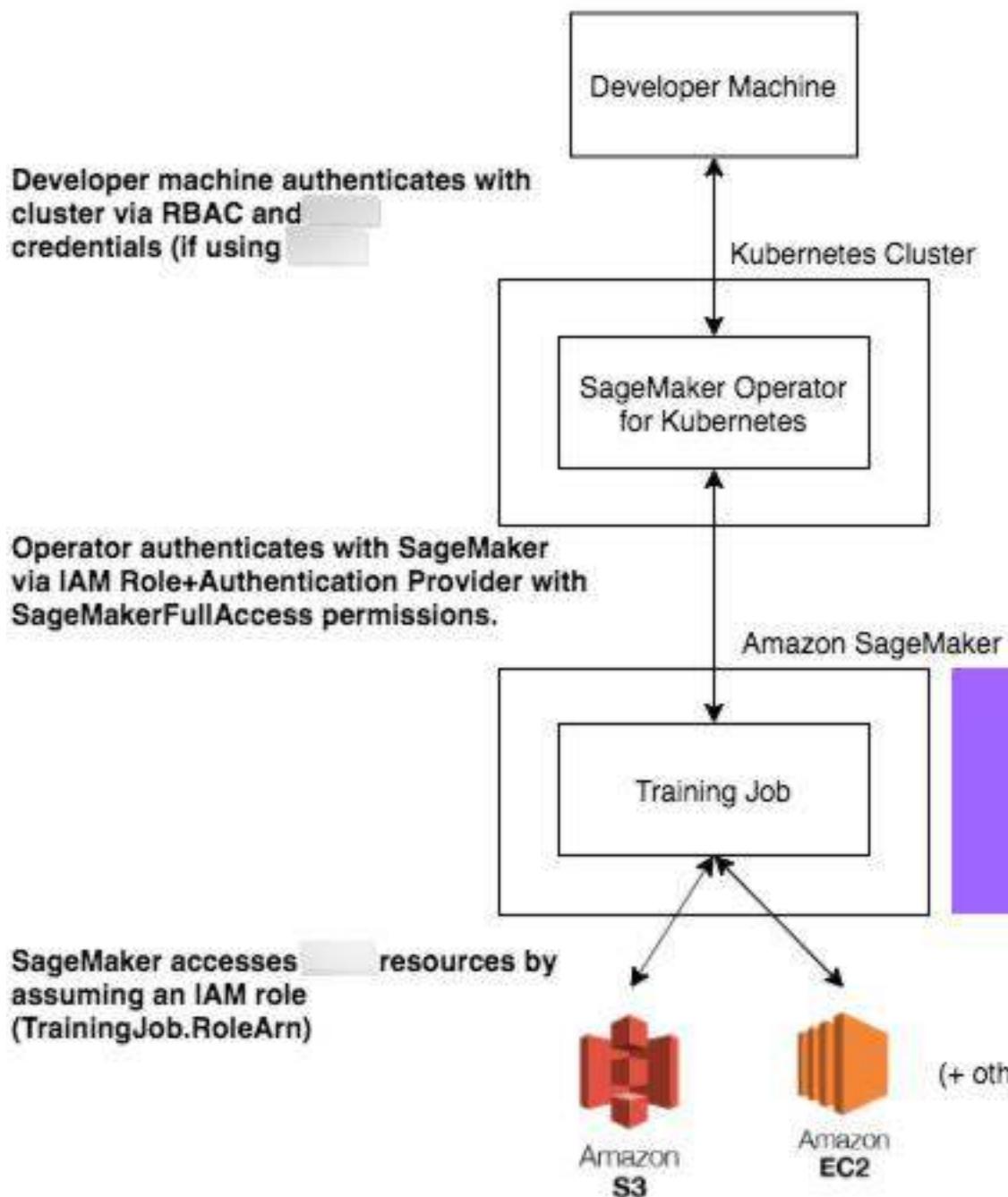
This guide assumes that you've completed the following prerequisites:

- Installed the following tools on the client machine used to access your Kubernetes cluster:
  - `kubectl` Version 1.13 or later. Use a `kubectl` version that is within one minor version of your Amazon EKS cluster control plane. For example, a 1.13 `kubectl` client works with Kubernetes 1.13 and 1.14 clusters. OpenID Connect (OIDC) is not supported in versions earlier than 1.13.
  - `eksctl` Version 0.7.0 or later
  - [Amazon CLI](#) Version 1.16.232 or later
  - (optional) [Helm](#) Version 3.0 or later
  - [aws-iam-authenticator](#)
- Have IAM permissions to create roles and attach policies to roles.
- Created a Kubernetes cluster on which to run the operators. It should either be Kubernetes version 1.13 or 1.14. For automated cluster creation using `eksctl`, see [Getting Started with eksctl](#). It takes 20 to 30 minutes to provision a cluster.

## Permissions overview

The SageMaker Operators for Kubernetes allow you to manage jobs in SageMaker from your Kubernetes cluster. The operators access SageMaker resources on your behalf. The IAM role that the operator assumes to interact with Amazon resources differs from the credentials you use to access the Kubernetes cluster. The role also differs from the role that SageMaker assumes when running your machine learning jobs. The following image explains this design and flow.

## Authentication Layers in the SageMaker Operator for Kubernetes



### IAM role-based setup and operator deployment

The following sections describe the steps to set up and deploy the operator.

## Cluster-scoped deployment

Before you can deploy your operator using an IAM role, associate an OpenID Connect (OIDC) provider with your role to authenticate with the IAM service.

### Create an OpenID Connect Provider for Your Cluster

The following instructions show how to create and associate an OIDC provider with your Amazon EKS cluster.

1. Set the local CLUSTER\_NAME and AWS\_REGION environment variables as follows:

```
# Set the Region and cluster
export CLUSTER_NAME="<your cluster name>"
export Amazon_REGION="<your region>"
```

2. Use the following command to associate the OIDC provider with your cluster. For more information, see [Enabling IAM Roles for Service Accounts on your Cluster](#).

```
eksctl utils associate-iam-oidc-provider --cluster ${CLUSTER_NAME} \
--region ${AWS_REGION} --approve
```

Your output should look like the following:

```
[__] eksctl version 0.10.1
[__] using region us-east-1
[__] IAM OpenID Connect provider is associated with cluster "my-cluster" in "us-east-1"
```

Now that the cluster has an OIDC identity provider, you can create a role and give a Kubernetes ServiceAccount permission to assume the role.

### Get the OIDC ID

To set up the ServiceAccount, obtain the OpenID Connect issuer URL using the following command:

```
aws eks describe-cluster --name ${CLUSTER_NAME} --region ${AWS_REGION} \
--query cluster.identity.oidc.issuer --output text
```

The command returns a URL like the following:

```
https://oidc.eks.us-east-1.amazonaws.com/id/D48675832CA65BD10A532F597OIDCID
```

In this URL, the value D48675832CA65BD10A532F597OIDCID is the OIDC ID. The OIDC ID for your cluster is different. You need this OIDC ID value to create a role.

If your output is None, it means that your client version is old. To work around this, run the following command:

```
aws eks describe-cluster --region ${AWS_REGION} --query cluster --name ${CLUSTER_NAME} --output text | grep OIDC
```

The OIDC URL is returned as follows:

```
OIDC https://oidc.eks.us-east-1.amazonaws.com/id/D48675832CA65BD10A532F597OIDCID
```

## Create an IAM Role

1. Create a file named `trust.json` and insert the following trust relationship code block into it instead. Be sure to replace all `<OIDC ID>`, `<Amazon account number>`, and `<EKS Cluster region>` placeholders with values corresponding to your cluster.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws-cn:iam::<Amazon account number>:oidc-provider/
oidc.eks.<EKS Cluster region>.amazonaws.com/id/<OIDC ID>"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.<EKS Cluster region>.amazonaws.com/id/<OIDC ID>:aud": "sts.amazonaws.com",
                    "oidc.eks.<EKS Cluster region>.amazonaws.com/id/<OIDC ID>:sub": "system:serviceaccount:sagemaker-k8s-operator-system:sagemaker-k8s-operator-default"
                }
            }
        }
    ]
}
```

2. Run the following command to create a role with the trust relationship defined in `trust.json`. This role enables the Amazon EKS cluster to get and refresh credentials from IAM.

```
aws iam create-role --region ${AWS_REGION} --role-name <role name> --assume-role-
policy-document file://trust.json --output=text
```

Your output should look like the following:

```
ROLE      arn:aws:iam::123456789012:role/my-role 2019-11-22T21:46:10Z      /
  ABCDEF0DNN7EXAMPLE      my-role
ASSUMEROLEPOLICYDOCUMENT      2012-10-17
STATEMENT      sts:AssumeRoleWithWebIdentity      Allow
STRINGEQUALS      sts.amazonaws.com      system:serviceaccount:sagemaker-k8s-operator-
system:sagemaker-k8s-operator-default
PRINCIPAL      arn:aws:iam::123456789012:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/
```

Take note of `ROLE ARN`; you pass this value to your operator.

## Attach the `AmazonSageMakerFullAccess` Policy to the Role

To give the role access to SageMaker, attach the `AmazonSageMakerFullAccess` policy. If you want to limit permissions to the operator, you can create your own custom policy and attach it.

Attach the following policy:

```
aws iam attach-role-policy --region ${AWS_REGION} --role-name <role name> --policy-arn
arn:aws-cn:iam::aws:policy/AmazonSageMakerFullAccess
```

The Kubernetes ServiceAccount `sagemaker-k8s-operator-default` should have `AmazonSageMakerFullAccess` permissions. Confirm this when you install the operator.

## Deploy the Operator

When deploying your operator, you can use either a YAML file or Helm charts.

### Deploy the Operator Using YAML

This is the simplest way to deploy your operators. The process is as follows:

1. Download the following installer script. Whenever `installer.yaml` is referenced, use `installer_china.yaml` instead.

```
wget https://raw.githubusercontent.com/aws/amazon-sagemaker-operator-for-k8s/master/release/rolebased/china/installer_china.yaml
```

2. Edit the `installer.yaml` file to replace `eks.amazonaws.com/role-arn`. Replace the ARN here with the Amazon Resource Name (ARN) for the OIDC-based role you've created.
3. Use the following command to deploy the cluster:

```
kubectl apply -f installer.yaml
```

### Deploy the Operator Using Helm Charts

Use the provided Helm Chart to install the operator.

1. Clone the Helm installer directory using the following command:

```
git clone https://github.com/aws/amazon-sagemaker-operator-for-k8s.git
```

2. Navigate to the `amazon-sagemaker-operator-for-k8s/hack/charts/installer` folder. Edit the `rolebased/values.yaml` file, which includes high-level parameters for the chart. Replace the role ARN here with the Amazon Resource Name (ARN) for the OIDC-based role you've created.
3. Install the Helm Chart using the following command:

```
kubectl create namespace sagemaker-k8s-operator-system
helm install --namespace sagemaker-k8s-operator-system sagemaker-operator rolebased/
```

If you decide to install the operator into a namespace other than the one specified, you need to adjust the namespace defined in the IAM role `trust.json` file to match.

4. After a moment, the chart is installed with a randomly generated name. Verify that the installation succeeded by running the following command:

```
helm ls
```

Your output should look like the following:

NAME	NAMESPACE	STATUS	CHART	REVISION	UPDATED	APP VERSION
sagemaker-operator	sagemaker-k8s-operator-system	23:14:59.6777082 +0000 UTC	deployed	1	2019-11-20	sagemaker-k8s-operator-0.1.0

### Verify the operator deployment

1. You should be able to see the SageMaker Custom Resource Definitions (CRDs) for each operator deployed to your cluster by running the following command:

```
kubectl get crd | grep sagemaker
```

Your output should look like the following:

batchtransformjobs.sagemaker.aws.amazon.com	2019-11-20T17:12:34Z
endpointconfigs.sagemaker.aws.amazon.com	2019-11-20T17:12:34Z
hostingdeployments.sagemaker.aws.amazon.com	2019-11-20T17:12:34Z
hyperparametertuningjobs.sagemaker.aws.amazon.com	2019-11-20T17:12:34Z
models.sagemaker.aws.amazon.com	2019-11-20T17:12:34Z
trainingjobs.sagemaker.aws.amazon.com	2019-11-20T17:12:34Z

2. Ensure that the operator pod is running successfully. Use the following command to list all pods:

```
kubectl -n sagemaker-k8s-operator-system get pods
```

You should see a pod named `sagemaker-k8s-operator-controller-manager-*****` in the namespace `sagemaker-k8s-operator-system` as follows:

NAME	READY	STATUS	RESTARTS
AGE			
sagemaker-k8s-operator-controller-manager-12345678-r8abc	2/2	Running	0
23s			

## Namespace-scoped deployment

You have the option to install your operator within the scope of an individual Kubernetes namespace. In this mode, the controller only monitors and reconciles resources with SageMaker if the resources are created within that namespace. This allows for finer-grained control over which controller is managing which resources. This is useful for deploying to multiple Amazon accounts or controlling which users have access to particular jobs.

This guide outlines how to install an operator into a particular, predefined namespace. To deploy a controller into a second namespace, follow the guide from beginning to end and change out the namespace in each step.

### Create an OpenID Connect Provider for Your Amazon EKS cluster

The following instructions show how to create and associate an OIDC provider with your Amazon EKS cluster.

1. Set the local `CLUSTER_NAME` and `AWS_REGION` environment variables as follows:

```
# Set the region and cluster
export CLUSTER_NAME="<your cluster name>"
export Amazon_REGION="<your region>"
```

2. Use the following command to associate the OIDC provider with your cluster. For more information, see [Enabling IAM Roles for Service Accounts on your Cluster](#).

```
eksctl utils associate-iam-oidc-provider --cluster ${CLUSTER_NAME} \
--region ${AWS_REGION} --approve
```

Your output should look like the following:

```
[__] eksctl version 0.10.1
```

```
[_] using region us-east-1
[_] IAM OpenID Connect provider is associated with cluster "my-cluster" in "us-east-1"
```

Now that the cluster has an OIDC identity provider, create a role and give a Kubernetes ServiceAccount permission to assume the role.

### [Get your OIDC ID](#)

To set up the ServiceAccount, first obtain the OpenID Connect issuer URL using the following command:

```
aws eks describe-cluster --name ${CLUSTER_NAME} --region ${AWS_REGION} \
--query cluster.identity.oidc.issuer --output text
```

The command returns a URL like the following:

```
https://oidc.eks.${AWS_REGION}.amazonaws.com/id/D48675832CA65BD10A532F597OIDCID
```

In this URL, the value D48675832CA65BD10A532F597OIDCID is the OIDC ID. The OIDC ID for your cluster will be different. You need this OIDC ID value to create a role.

If your output is None, it means that your client version is old. To work around this, run the following command:

```
aws eks describe-cluster --region ${AWS_REGION} --query cluster --name ${CLUSTER_NAME} --output text | grep OIDC
```

The OIDC URL is returned as follows:

```
OIDC https://oidc.eks.us-east-1.amazonaws.com/id/D48675832CA65BD10A532F597OIDCID
```

### [Create your IAM Role](#)

1. Create a file named `trust.json` and insert the following trust relationship code block into it instead. Be sure to replace all <OIDC ID>, <Amazon account number>, and <EKS Cluster region> placeholders with values corresponding to your cluster.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws-cn:iam::<Amazon account number>:oidc-provider/
oidc.eks.<EKS Cluster region>.amazonaws.com/id/<OIDC ID>"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.<EKS Cluster region>.amazonaws.com/id/<OIDC ID>:aud": "sts.amazonaws.com",
                    "oidc.eks.<EKS Cluster region>.amazonaws.com/id/<OIDC ID>:sub": "system:serviceaccount:<Namespace>:sagemaker-k8s-operator-default"
                }
            }
        ]
    ]
}
```

- Run the following command to create a role with the trust relationship defined in `trust.json`. This role enables the Amazon EKS cluster to get and refresh credentials from IAM.

```
aws iam create-role --region ${AWS_REGION} --role-name <role name> --assume-role-policy-document file://trust.json --output=text
```

Your output should look like the following:

```
ROLE      arn:aws:iam::123456789012:role/my-role 2019-11-22T21:46:10Z      /
  ABCDEFSFODNN7EXAMPLE      my-role
ASSUMEROLEPOLICYDOCUMENT      2012-10-17
STATEMENT      sts:AssumeRoleWithWebIdentity      Allow
STRINGEQUALS      sts.amazonaws.com      system:serviceaccount:my-namespace:sagemaker-
k8s-operator-default
PRINCIPAL      arn:aws:iam::123456789012:oidc-provider/oidc.eks.us-
east-1.amazonaws.com/id/
```

Take note of `ROLE ARN`. You pass this value to your operator.

### Attach the `AmazonSageMakerFullAccess` Policy to your Role

To give the role access to SageMaker, attach the [AmazonSageMakerFullAccess](#) policy. If you want to limit permissions to the operator, you can create your own custom policy and attach it.

Attach the following policy:

```
aws iam attach-role-policy --region ${AWS_REGION} --role-name <role name> --policy-arn
arn:aws-cn:iam::aws:policy/AmazonSageMakerFullAccess
```

The Kubernetes ServiceAccount `sagemaker-k8s-operator-default` should have `AmazonSageMakerFullAccess` permissions. Confirm this when you install the operator.

### Deploy the Operator to Your Namespace

When deploying your operator, you can use either a YAML file or Helm charts.

#### Deploy the Operator to Your Namespace Using YAML

There are two parts to deploying an operator within the scope of a namespace. The first is the set of CRDs that are installed at a cluster level. These resource definitions only need to be installed once per Kubernetes cluster. The second part is the operator permissions and deployment itself.

If you have not already installed the CRDs into the cluster, apply the CRD installer YAML using the following command:

```
kubectl apply -f https://raw.githubusercontent.com/aws/amazon-sagemaker-operator-for-k8s/
master/release/rolebased/namespaced/crd.yaml
```

To install the operator onto the cluster:

- Download the following operator script. Whenever `operator.yaml` is referenced, use `operator_china.yaml` instead.

```
wget https://raw.githubusercontent.com/aws/amazon-sagemaker-operator-for-k8s/master/
release/rolebased/namespaced/china/operator_china.yaml
```

- Update the installer YAML to place the resources into your specified namespace using the following command:

```
sed -i -e 's/PLACEHOLDER-NAMESPACE/<YOUR NAMESPACE>/g' operator.yaml
```

3. Edit the `operator.yaml` file to place resources into your `eks.amazonaws.com/role-arn`. Replace the ARN here with the Amazon Resource Name (ARN) for the OIDC-based role you've created.
4. Use the following command to deploy the cluster:

```
kubectl apply -f operator.yaml
```

## Deploy the Operator to Your Namespace Using Helm Charts

There are two parts needed to deploy an operator within the scope of a namespace. The first is the set of CRDs that are installed at a cluster level. These resource definitions only need to be installed once per Kubernetes cluster. The second part is the operator permissions and deployment itself. When using helm charts you have to first create the namespace using `kubectl`.

1. Clone the Helm installer directory using the following command:

```
git clone https://github.com/aws/amazon-sagemaker-operator-for-k8s.git
```

2. Navigate to the `amazon-sagemaker-operator-for-k8s/hack/charts/installer/namespaced` folder. Edit the `rolebased/values.yaml` file, which includes high-level parameters for the chart. Replace the role ARN here with the Amazon Resource Name (ARN) for the OIDC-based role you've created.
3. Install the Helm Chart using the following command:

```
helm install crds crd_chart/
```

4. Create the required namespace and install the operator using the following command:

```
kubectl create namespace <namespace>
helm install --n <namespace> op operator_chart/
```

5. After a moment, the chart is installed with the name `sagemaker-operator`. Verify that the installation succeeded by running the following command:

```
helm ls
```

Your output should look like the following:

NAME	NAMESPACE	STATUS	CHART	REVISION	UPDATED
					APP VERSION
sagemaker-operator	my-namespace			1	2019-11-20
23:14:59.6777082 +0000 UTC	deployed				sagemaker-k8s-operator-0.1.0

## Verify the operator deployment to your namespace

1. You should be able to see the SageMaker Custom Resource Definitions (CRDs) for each operator deployed to your cluster by running the following command:

```
kubectl get crd | grep sagemaker
```

Your output should look like the following:

batchtransformjobs.sagemaker.aws.amazon.com	2019-11-20T17:12:34Z
endpointconfigs.sagemaker.aws.amazon.com	2019-11-20T17:12:34Z
hostingdeployments.sagemaker.aws.amazon.com	2019-11-20T17:12:34Z
hyperparametertuningjobs.sagemaker.aws.amazon.com	2019-11-20T17:12:34Z
models.sagemaker.aws.amazon.com	2019-11-20T17:12:34Z
trainingjobs.sagemaker.aws.amazon.com	2019-11-20T17:12:34Z

2. Ensure that the operator pod is running successfully. Use the following command to list all pods:

```
kubectl -n my-namespace get pods
```

You should see a pod named `sagemaker-k8s-operator-controller-manager-*****` in the namespace `my-namespace` as follows:

NAME	READY	STATUS	RESTARTS
AGE sagemaker-k8s-operator-controller-manager-12345678-r8abc 23s	2/2	Running	0

## Install the SageMaker logs `kubectl` plugin

As part of the SageMaker Operators for Kubernetes, you can use the [smlogs plugin](#) for `kubectl`. This enables SageMaker CloudWatch logs to be streamed with `kubectl`. `kubectl` must be installed onto your [PATH](#). The following commands place the binary in the `sagemaker-k8s-bin` directory in your home directory, and add that directory to your [PATH](#).

```
export os="linux"

wget https://amazon-sagemaker-operator-for-k8s-us-east-1.s3.amazonaws.com/kubectl-smlogs-
plugin/v1/${os}.amd64.tar.gz
tar xvzf ${os}.amd64.tar.gz

# Move binaries to a directory in your homedir.
mkdir ~/.sagemaker-k8s-bin
cp ./kubectl-smlogs.${os}.amd64/kubectl-smlogs ~/.sagemaker-k8s-bin/.

# This line adds the binaries to your PATH in your .bashrc.

echo 'export PATH=$PATH:~/sagemaker-k8s-bin' >> ~/.bashrc

# Source your .bashrc to update environment variables:
source ~/.bashrc
```

Use the following command to verify that the `kubectl` plugin is installed correctly:

```
kubectl smlogs
```

If the `kubectl` plugin is installed correctly, your output should look like the following:

```
View SageMaker logs via Kubernetes

Usage:
  smlogs [command]
```

```
Aliases:  
  smlogs, SMLogs, Smlogs  
  
Available Commands:  
  BatchTransformJob      View BatchTransformJob logs via Kubernetes  
  TrainingJob           View TrainingJob logs via Kubernetes  
  help                  Help about any command  
  
Flags:  
  -h, --help    help for smlogs  
  
Use "smlogs [command] --help" for more information about a command.
```

## Delete operators

### Delete cluster-based operators

#### Operators installed using YAML

To uninstall the operator from your cluster, make sure that all SageMaker resources have been deleted from the cluster. Failure to do so causes the operator delete operation to hang. Once you have deleted all SageMaker jobs, use `kubectl` to delete the operator from the cluster. Run the following commands to stop all jobs and delete the operator from the cluster:

```
# Delete all SageMaker jobs from Kubernetes  
kubectl delete --all --all-namespaces hyperparametertuningjob.sagemaker.aws.amazon.com  
kubectl delete --all --all-namespaces trainingjobs.sagemaker.aws.amazon.com  
kubectl delete --all --all-namespaces batchtransformjob.sagemaker.aws.amazon.com  
kubectl delete --all --all-namespaces hostingdeployment.sagemaker.aws.amazon.com  
  
# Delete the operator and its resources  
kubectl delete -f /installer.yaml
```

You should see output like the following:

```
$ kubectl delete --all --all-namespaces trainingjobs.sagemaker.aws.amazon.com  
trainingjobs.sagemaker.aws.amazon.com "xgboost-mnist-from-for-s3" deleted  
  
$ kubectl delete --all --all-namespaces hyperparametertuningjob.sagemaker.aws.amazon.com  
hyperparametertuningjob.sagemaker.aws.amazon.com "xgboost-mnist-hpo" deleted  
  
$ kubectl delete --all --all-namespaces batchtransformjob.sagemaker.aws.amazon.com  
batchtransformjob.sagemaker.aws.amazon.com "xgboost-mnist" deleted  
  
$ kubectl delete --all --all-namespaces hostingdeployment.sagemaker.aws.amazon.com  
hostingdeployment.sagemaker.aws.amazon.com "host-xgboost" deleted  
  
$ kubectl delete -f raw-yaml/installer.yaml  
namespace "sagemaker-k8s-operator-system" deleted  
customresourcedefinition.apiextensions.k8s.io "batchtransformjobs.sagemaker.aws.amazon.com"  
  deleted  
customresourcedefinition.apiextensions.k8s.io "endpointconfigs.sagemaker.aws.amazon.com"  
  deleted  
customresourcedefinition.apiextensions.k8s.io "hostingdeployments.sagemaker.aws.amazon.com"  
  deleted  
customresourcedefinition.apiextensions.k8s.io  
  "hyperparametertuningjobs.sagemaker.aws.amazon.com" deleted  
customresourcedefinition.apiextensions.k8s.io "models.sagemaker.aws.amazon.com" deleted  
customresourcedefinition.apiextensions.k8s.io "trainingjobs.sagemaker.aws.amazon.com"  
  deleted  
role.rbac.authorization.k8s.io "sagemaker-k8s-operator-leader-election-role" deleted
```

```

clusterrole.rbac.authorization.k8s.io "sagemaker-k8s-operator-manager-role" deleted
clusterrole.rbac.authorization.k8s.io "sagemaker-k8s-operator-proxy-role" deleted
rolebinding.rbac.authorization.k8s.io "sagemaker-k8s-operator-leader-election-rolebinding"
    deleted
clusterrolebinding.rbac.authorization.k8s.io "sagemaker-k8s-operator-manager-rolebinding"
    deleted
clusterrolebinding.rbac.authorization.k8s.io "sagemaker-k8s-operator-proxy-rolebinding"
    deleted
service "sagemaker-k8s-operator-controller-metrics-service" deleted
deployment.apps "sagemaker-k8s-operator-controller-manager" deleted
secrets "sagemaker-k8s-operator-abcde" deleted

```

## Operators installed using Helm Charts

To delete the operator CRDs, first delete all the running jobs. Then delete the Helm Chart that was used to deploy the operators using the following commands:

```

# get the helm charts
$ helm ls

# delete the charts
$ helm delete <chart name>

```

## Delete namespace-based operators

### Operators installed with YAML

To uninstall the operator from your cluster, make sure that all SageMaker resources have been deleted from the cluster. Failure to do so causes the operator delete operation to hang. Once you have deleted all SageMaker jobs, use kubectl to first delete the operator from the namespace and then the CRDs from the cluster. Run the following commands to stop all jobs and delete the operator from the cluster:

```

# Delete all SageMaker jobs from Kubernetes
kubectl delete --all --all-namespaces hyperparametertuningjob.sagemaker.aws.amazon.com
kubectl delete --all --all-namespaces trainingjobs.sagemaker.aws.amazon.com
kubectl delete --all --all-namespaces batchtransformjob.sagemaker.aws.amazon.com
kubectl delete --all --all-namespaces hostingdeployment.sagemaker.aws.amazon.com

```

```

# Delete the operator using the same yaml file that was used to install the operator
kubectl delete -f operator.yaml

# Now delete the CRDs using the CRD installer yaml
kubectl delete -f https://raw.githubusercontent.com/aws/amazon-sagemaker-operator-for-k8s/
master/release/rolebased/namespaced/crd.yaml

# Now you can delete the namespace if you want
kubectl delete namespace <namespace>

```

### Operators installed with Helm Charts

To delete the operator CRDs, first delete all the running jobs. Then delete the Helm Chart that was used to deploy the operators using the following commands:

```

# Delete the operator
$ helm delete -n <namespace> op

# delete the crds
$ helm delete crds

```

```
# optionally delete the namespace
$ kubectl delete namespace <namespace>
```

## Troubleshooting

### Debugging a Failed Job

- Check the job status by running the following:

```
kubectl get <CRD Type> <job name>
```

- If the job was created in SageMaker, you can use the following command to see the STATUS and the SageMaker Job Name:

```
kubectl get <crd type> <job name>
```

- You can use smlogs to find the cause of the issue using the following command:

```
kubectl smlogs <crd type> <job name>
```

- You can also use the describe command to get more details about the job using the following command. The output has an additional field that has more information about the status of the job.

```
kubectl describe <crd type> <job name>
```

- If the job was not created in SageMaker, then use the logs of the operator's pod to find the cause of the issue as follows:

```
$ kubectl get pods -A | grep sagemaker
# Output:
sagemaker-k8s-operator-system    sagemaker-k8s-operator-controller-manager-5cd7df4d74-
wh22z    2/2      Running     0          3h33m

$ kubectl logs -p <pod name> -c manager -n sagemaker-k8s-operator-system
```

### Deleting an Operator CRD

If deleting a job is not working, check if the operator is running. If the operator is not running, then you have to delete the finalizer using the following steps:

1. In a new terminal, open the job in an editor using `kubectl edit` as follows:

```
$ kubectl edit <crd type> <job name>
```

2. Edit the job to delete the finalizer by removing the following two lines from the file. Save the file and the job is be deleted.

```
finalizers:
- sagemaker-operator-finalizer
```

## Images and SMlogs in each Region

The following table lists the available operator images and SMLogs in each region.

Region	Controller Image	Linux SMLogs
us-east-1	957583890962.dkr.ecr.us-east-1.amazonaws.com/amazon-sagemaker-operator-for-k8s:v1	<a href="https://s3.us-east-1.amazonaws.com/amazon-sagemaker-operator-for-k8s-us-east-1/kubectl-smlogs-plugin/v1/linux.amd64.tar.gz">https://s3.us-east-1.amazonaws.com/amazon-sagemaker-operator-for-k8s-us-east-1/kubectl-smlogs-plugin/v1/linux.amd64.tar.gz</a>
us-east-2	922499468684.dkr.ecr.us-east-2.amazonaws.com/amazon-sagemaker-operator-for-k8s:v1	<a href="https://s3.us-east-2.amazonaws.com/amazon-sagemaker-operator-for-k8s-us-east-2/kubectl-smlogs-plugin/v1/linux.amd64.tar.gz">https://s3.us-east-2.amazonaws.com/amazon-sagemaker-operator-for-k8s-us-east-2/kubectl-smlogs-plugin/v1/linux.amd64.tar.gz</a>
us-west-2	640106867763.dkr.ecr.us-west-2.amazonaws.com/amazon-sagemaker-operator-for-k8s:v1	<a href="https://s3.us-west-2.amazonaws.com/amazon-sagemaker-operator-for-k8s-us-west-2/kubectl-smlogs-plugin/v1/linux.amd64.tar.gz">https://s3.us-west-2.amazonaws.com/amazon-sagemaker-operator-for-k8s-us-west-2/kubectl-smlogs-plugin/v1/linux.amd64.tar.gz</a>
eu-west-1	613661167059.dkr.ecr.eu-west-1.amazonaws.com/amazon-sagemaker-operator-for-k8s:v1	<a href="https://s3.eu-west-1.amazonaws.com/amazon-sagemaker-operator-for-k8s-eu-west-1/kubectl-smlogs-plugin/v1/linux.amd64.tar.gz">https://s3.eu-west-1.amazonaws.com/amazon-sagemaker-operator-for-k8s-eu-west-1/kubectl-smlogs-plugin/v1/linux.amd64.tar.gz</a>

## Using SageMaker Jobs

To run a job using the SageMaker Operators for Kubernetes, you can either apply a YAML file or use the supplied Helm Charts.

All operator sample jobs in the following tutorials use sample data taken from a public MNIST dataset. In order to run these samples, download the dataset into your Amazon S3 bucket. You can find the dataset in [Download the MNIST Dataset](#).

### Contents

- [TrainingJob operator \(p. 2277\)](#)
- [HyperParameterTuningJobs operator \(p. 2281\)](#)
- [BatchTransformJobs operator \(p. 2286\)](#)
- [Real-time inference \(p. 674\)](#)

## TrainingJob operator

Training job operators reconcile your specified training job spec to SageMaker by launching it for you in SageMaker. You can learn more about SageMaker training jobs in the SageMaker [CreateTrainingJob API documentation](#).

### Create a TrainingJob Using a Simple YAML File

1. Download the sample YAML file for training using the following command:

```
wget https://raw.githubusercontent.com/aws/amazon-sagemaker-operator-for-k8s/master/
samples/xgboost-mnist-trainingjob.yaml
```

2. Edit the `xgboost-mnist-trainingjob.yaml` file to replace the `roleArn` parameter with your `<sagemaker-execution-role>`, and `outputPath` with your Amazon S3 bucket that the SageMaker execution role has write access to. The `roleArn` must have permissions so that SageMaker can access Amazon S3, Amazon CloudWatch, and other services on your behalf. For more information on creating an SageMaker ExecutionRole, see [SageMaker Roles](#). Apply the YAML file using the following command:

```
kubectl apply -f xgboost-mnist-trainingjob.yaml
```

## Create a TrainingJob Using a Helm Chart

You can use Helm Charts to run TrainingJobs.

1. Clone the GitHub repo to get the source using the following command:

```
git clone https://github.com/aws/amazon-sagemaker-operator-for-k8s.git
```

2. Navigate to the `amazon-sagemaker-operator-for-k8s/hack/charts/training-jobs/` folder and edit the `values.yaml` file to replace values like `rolearn` and `outputpath` with values that correspond to your account. The RoleARN must have permissions so that SageMaker can access Amazon S3, Amazon CloudWatch, and other services on your behalf. For more information on creating an SageMaker ExecutionRole, see [SageMaker Roles](#).

## Create the Training Job

With the roles and Amazon S3 buckets replaced with appropriate values in `values.yaml`, you can create a training job using the following command:

```
helm install . --generate-name
```

Your output should look like the following:

```
NAME: chart-12345678
LAST DEPLOYED: Wed Nov 20 23:35:49 2019
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thanks for installing the sagemaker-k8s-trainingjob.
```

## Verify Your Training Helm Chart

To verify that the Helm Chart was created successfully, run:

```
helm ls
```

Your output should look like the following:

NAME	STATUS	NAMESPACE	REVISION	UPDATED	APP VERSION
		CHART			
chart-12345678	deployed	default	1	2019-11-20 23:35:49.9136092 +0000 UTC	sagemaker-k8s-trainingjob-0.1.0
rolebased-12345678	deployed	default	1	2019-11-20 23:14:59.6777082 +0000 UTC	sagemaker-k8s-operator-0.1.0

`helm install` creates a TrainingJob Kubernetes resource. The operator launches the actual training job in SageMaker and updates the TrainingJob Kubernetes resource to reflect the status of the job in SageMaker. You incur charges for SageMaker resources used during the duration of your job. You do not incur any charges once your job completes or stops.

**Note:** SageMaker does not allow you to update a running training job. You cannot edit any parameter and re-apply the file/config. Either change the metadata name or delete the existing job and create a new one. Similar to existing training job operators like TFJob in Kubeflow, update is not supported.

## List Training Jobs

Use the following command to list all jobs created using the Kubernetes operator:

```
kubectl get TrainingJob
```

The output listing all jobs should look like the following:

```
kubectl get trainingjobs
NAME           STATUS    SECONDARY-STATUS   CREATION-TIME
SAGEMAKER-JOB-NAME
xgboost-mnist-from-for-s3   InProgress   Starting          2019-11-20T23:42:35Z   xgboost-
mnist-from-for-s3-examplef11eab94e0ed4671d5a8f
```

A training job continues to be listed after the job has completed or failed. You can remove a `TrainingJob` job from the list by following the Delete a Training Job steps. Jobs that have completed or stopped do not incur any charges for SageMaker resources.

### Training Job Status Values

The `STATUS` field can be one of the following values:

- `Completed`
- `InProgress`
- `Failed`
- `Stopped`
- `Stopping`

These statuses come directly from the SageMaker official [API documentation](#).

In addition to the official SageMaker status, it is possible for `STATUS` to be `SynchronizingK8sJobWithSageMaker`. This means that the operator has not yet processed the job.

### Secondary Status Values

The secondary statuses come directly from the SageMaker official [API documentation](#). They contain more granular information about the status of the job.

### Describe a Training Job

You can get more details about the training job by using the `describe kubectl` verb. This is typically used for debugging a problem or checking the parameters of a training job. To get information about your training job, use the following command:

```
kubectl describe trainingjob xgboost-mnist-from-for-s3
```

The output for your training job should look like the following:

```
Name:           xgboost-mnist-from-for-s3
Namespace:      default
Labels:         <none>
Annotations:   <none>
API Version:  sagemaker.aws.amazon.com/v1
Kind:          TrainingJob
Metadata:
  Creation Timestamp: 2019-11-20T23:42:35Z
Finalizers:
```

```

sagemaker-operator-finalizer
Generation: 2
Resource Version: 23119
Self Link: /apis/sagemaker.aws.amazon.com/v1/namespaces/default/trainingjobs/xgboost-mnist-from-for-s3
UID: 6d7uiui-0bef-11ea-b94e-0ed467example
Spec:
  Algorithm Specification:
    Training Image: 8256416981234.dkr.ecr.us-east-2.amazonaws.com/xgboost:1
    Training Input Mode: File
  Hyper Parameters:
    Name: eta
    Value: 0.2
    Name: gamma
    Value: 4
    Name: max_depth
    Value: 5
    Name: min_child_weight
    Value: 6
    Name: num_class
    Value: 10
    Name: num_round
    Value: 10
    Name: objective
    Value: multi:softmax
    Name: silent
    Value: 0
  Input Data Config:
    Channel Name: train
    Compression Type: None
    Content Type: text/csv
    Data Source:
      S 3 Data Source:
        S 3 Data Distribution Type: FullyReplicated
        S 3 Data Type: S3Prefix
        S 3 Uri: https://s3-us-east-2.amazonaws.com/my-bucket/sagemaker/xgboost-mnist/train/
    Channel Name: validation
    Compression Type: None
    Content Type: text/csv
    Data Source:
      S 3 Data Source:
        S 3 Data Distribution Type: FullyReplicated
        S 3 Data Type: S3Prefix
        S 3 Uri: https://s3-us-east-2.amazonaws.com/my-bucket/sagemaker/xgboost-mnist/validation/
  Output Data Config:
    S 3 Output Path: s3://my-bucket/sagemaker/xgboost-mnist/xgboost/
  Region: us-east-2
  Resource Config:
    Instance Count: 1
    Instance Type: ml.m4.xlarge
    Volume Size In GB: 5
    Role Arn: arn:aws:iam::12345678910:role/service-role/AmazonSageMaker-ExecutionRole
  Stopping Condition:
    Max Runtime In Seconds: 86400
  Training Job Name: xgboost-mnist-from-for-s3-6d7fa0af0bef11eab94e0example
Status:
  Cloud Watch Log URL: https://us-east-2.console.aws.amazon.com/cloudwatch/home?region=us-east-2#logStream:group=/aws/sagemaker/TrainingJobs;prefix=<example>;streamFilter=typeLogStreamPrefix
  Last Check Time: 2019-11-20T23:44:29Z
  Sage Maker Training Job Name: xgboost-mnist-from-for-s3-6d7fa0af0bef11eab94e0example
  Secondary Status: Downloading
  Training Job Status: InProgress

```

Events:	<none>
---------	--------

## View Logs from Training Jobs

Use the following command to see the logs from the kmeans-mnist training job:

```
kubectl smlogs trainingjob xgboost-mnist-from-for-s3
```

Your output should look similar to the following. The logs from instances are ordered chronologically.

```
"xgboost-mnist-from-for-s3" has SageMaker TrainingJobName "xgboost-mnist-from-for-s3-123456789" in region "us-east-2", status "InProgress" and secondary status "Starting" 23:45:24.7 +0000 UTC Arguments: train xgboost-mnist-from-for-s3-6d7fa0af0bef11eab94e0ed46example/algo-1-1574293123 2019-11-20 23:45:24.7 +0000 UTC [2019-11-20:23:45:22:INFO] Running standalone xgboost training. xgboost-mnist-from-for-s3-6d7fa0af0bef11eab94e0ed46example/algo-1-1574293123 2019-11-20 23:45:24.7 +0000 UTC [2019-11-20:23:45:22:INFO] File size need to be processed in the node: 1122.95mb. Available memory size in the node: 8586.0mb xgboost-mnist-from-for-s3-6d7fa0af0bef11eab94e0ed46example/algo-1-1574293123 2019-11-20 23:45:24.7 +0000 UTC [2019-11-20:23:45:22:INFO] Determined delimiter of CSV input is ',' xgboost-mnist-from-for-s3-6d7fa0af0bef11eab94e0ed46example/algo-1-1574293123 2019-11-20 23:45:24.7 +0000 UTC [23:45:22] S3DistributionType set as FullyReplicated
```

## Delete Training Jobs

Use the following command to stop a training job on Amazon SageMaker:

```
kubectl delete trainingjob xgboost-mnist-from-for-s3
```

This command removes the SageMaker training job from Kubernetes. This command returns the following output:

```
trainingjob.sagemaker.aws.amazon.com "xgboost-mnist-from-for-s3" deleted
```

If the job is still in progress on SageMaker, the job stops. You do not incur any charges for SageMaker resources after your job stops or completes.

**Note:** SageMaker does not delete training jobs. Stopped jobs continue to show on the SageMaker console. The delete command takes about 2 minutes to clean up the resources from SageMaker.

## HyperParameterTuningJobs operator

Hyperparameter tuning job operators reconcile your specified hyperparameter tuning job spec to SageMaker by launching it in SageMaker. You can learn more about SageMaker hyperparameter tuning jobs in the SageMaker [CreateHyperParameterTuningJob API documentation](#).

### Create a Hyperparameter Tuning Job Using a Simple YAML File

1. Download the sample YAML file for the hyperparameter tuning job using the following command:

```
wget https://raw.githubusercontent.com/aws/amazon-sagemaker-operator-for-k8s/master/samples/xgboost-mnist-hpo.yaml
```

2. Edit the `xgboost-mnist-hpo.yaml` file to replace the `roleArn` parameter with your `sagemaker-execution-role`. For the hyperparameter tuning job to succeed, you must also change the `s3InputPath` and `s3OutputPath` to values that correspond to your account. Apply the updates YAML file using the following command:

```
kubectl apply -f xgboost-mnist-hpo.yaml
```

## Create a Hyperparameter Tuning Job using a Helm Chart

You can use Helm Charts to run hyperparameter tuning jobs.

1. Clone the GitHub repo to get the source using the following command:

```
git clone https://github.com/aws/amazon-sagemaker-operator-for-k8s.git
```

2. Navigate to the `amazon-sagemaker-operator-for-k8s/hack/charts/hyperparameter-tuning-jobs/` folder.
3. Edit the `values.yaml` file to replace the `roleArn` parameter with your `sagemaker-execution-role`. For the hyperparameter tuning job to succeed, you must also change the `s3InputPath` and `s3OutputPath` to values that correspond to your account.

## Create the Hyperparameter Tuning Job

With the roles and Amazon S3 paths replaced with appropriate values in `values.yaml`, you can create a hyperparameter tuning job using the following command:

```
helm install . --generate-name
```

Your output should look similar to the following:

```
NAME: chart-1574292948
LAST DEPLOYED: Wed Nov 20 23:35:49 2019
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thanks for installing the sagemaker-k8s-hyperparametertuningjob.
```

## Verify Chart Installation

To verify that the Helm Chart was created successfully, run the following command:

```
helm ls
```

Your output should look like the following:

NAME	NAMESPACE	REVISION	UPDATED
chart-1474292948	default	1	2019-11-20 23:35:49.9136092 +0000
UTC deployed	sagemaker-k8s-hyperparametertuningjob-0.1.0		
STATUS	CHART		APP VERSION
chart-1574292948	default	1	2019-11-20 23:35:49.9136092 +0000
UTC deployed	sagemaker-k8s-trainingjob-0.1.0		
rolebased-1574291698	default	1	2019-11-20 23:14:59.6777082 +0000
UTC deployed	sagemaker-k8s-operator-0.1.0		

`helm install` creates a `HyperParameterTuningJob` Kubernetes resource. The operator launches the actual hyperparameter optimization job in SageMaker and updates the `HyperParameterTuningJob`

Kubernetes resource to reflect the status of the job in SageMaker. You incur charges for SageMaker resources used during the duration of your job. You do not incur any charges once your job completes or stops.

**Note:** SageMaker does not allow you to update a running hyperparameter tuning job. You cannot edit any parameter and re-apply the file/config. You must either change the metadata name or delete the existing job and create a new one. Similar to existing training job operators like TFJob in Kubeflow, update is not supported.

## List Hyperparameter Tuning Jobs

Use the following command to list all jobs created using the Kubernetes operator:

```
kubectl get hyperparametertuningjob
```

Your output should look like the following:

NAME	STATUS	CREATION-TIME	COMPLETED	INPROGRESS	ERRORS	STOPPED
SAGEMAKER-JOB-NAME						
BEST-TRAINING-JOB	Completed	2019-10-17T01:15:52Z	10	0	0	0
xgboost-mnist-hpo		xgboosttha92f5e3cf07b11e9bf6c06d6-009-4c7a123		xgboosttha92f5e3cf07b11e9bf6c123		

A hyperparameter tuning job continues to be listed after the job has completed or failed. You can remove a hyperparametertuningjob from the list by following the steps in [Delete a Hyperparameter Tuning Job](#). Jobs that have completed or stopped do not incur any charges for SageMaker resources.

## Hyperparameter Tuning Job Status Values

The STATUS field can be one of the following values:

- Completed
- InProgress
- Failed
- Stopped
- Stopping

These statuses come directly from the SageMaker official [API documentation](#).

In addition to the official SageMaker status, it is possible for STATUS to be SynchronizingK8sJobWithSageMaker. This means that the operator has not yet processed the job.

## Status Counters

The output has several counters, like COMPLETED and INPROGRESS. These represent how many training jobs have completed and are in progress, respectively. For more information about how these are determined, see [TrainingJobStatusCounters](#) in the SageMaker API documentation.

## Best Training Job

This column contains the name of the TrainingJob that best optimized the selected metric.

To see a summary of the tuned hyperparameters, run:

```
kubectl describe hyperparametertuningjob xgboost-mnist-hpo
```

To see detailed information about the TrainingJob, run:

```
kubectl describe trainingjobs <job name>
```

## Spawned Training Jobs

You can also track all 10 training jobs in Kubernetes launched by `HyperparameterTuningJob` by running the following command:

```
kubectl get trainingjobs
```

## Describe a Hyperparameter Tuning Job

You can obtain debugging details using the `describe` `kubectl` verb by running the following command.

```
kubectl describe hyperparametertuningjob xgboost-mnist-hpo
```

In addition to information about the tuning job, the SageMaker Operator for Kubernetes also exposes the [best training job](#) found by the hyperparameter tuning job in the `describe` output as follows:

```
Name:           xgboost-mnist-hpo
Namespace:      default
Labels:         <none>
Annotations:   kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"sagemaker.aws.amazon.com/v1","kind":"HyperparameterTuningJob","metadata":{"annotations":{},"name":"xgboost-mnist-hpo","namespace":...}}
API Version:   sagemaker.aws.amazon.com/v1
Kind:          HyperparameterTuningJob
Metadata:
  Creation Timestamp: 2019-10-17T01:15:52Z
  Finalizers:
    sagemaker-operator-finalizer
  Generation: 2
  Resource Version: 8167
  Self Link: /apis/sagemaker.aws.amazon.com/v1/namespaces/default/hyperparametertuningjobs/xgboost-mnist-hpo
  UID:        a92f5e3c-f07b-11e9-bf6c-06d6f303uidu
Spec:
  Hyper Parameter Tuning Job Config:
    Hyper Parameter Tuning Job Objective:
      Metric Name: validation:error
      Type:       Minimize
    Parameter Ranges:
      Integer Parameter Ranges:
        Max Value: 20
        Min Value: 10
        Name:       num_round
        Scaling Type: Linear
    Resource Limits:
      Max Number Of Training Jobs: 10
      Max Parallel Training Jobs: 10
    Strategy:      Bayesian
    Training Job Early Stopping Type: Off
  Hyper Parameter Tuning Job Name: xgboostha92f5e3cf07b11e9bf6c06d6
  Region:        us-east-2
  Training Job Definition:
    Algorithm Specification:
      Training Image: 12345678910.dkr.ecr.us-east-2.amazonaws.com/xgboost:1
      Training Input Mode: File
    Input Data Config:
      Channel Name: train
```

```

Content Type: text/csv
Data Source:
  s3DataSource:
    s3DataDistributionType: FullyReplicated
    s3DataType: S3Prefix
    s3Uri: https://s3-us-east-2.amazonaws.com/my-bucket/sagemaker/
xgboost-mnist/train/
  Channel Name: validation
  Content Type: text/csv
  Data Source:
    s3DataSource:
      s3DataDistributionType: FullyReplicated
      s3DataType: S3Prefix
      s3Uri: https://s3-us-east-2.amazonaws.com/my-bucket/sagemaker/
xgboost-mnist/validation/
  Output Data Config:
    s3OutputPath: https://s3-us-east-2.amazonaws.com/my-bucket/sagemaker/xgboost-mnist/
xgboost
  Resource Config:
    Instance Count: 1
    Instance Type: ml.m4.xlarge
    Volume Size In GB: 5
    Role Arn: arn:aws:iam::123456789012:role/service-role/AmazonSageMaker-
ExecutionRole
  Static Hyper Parameters:
    Name: base_score
    Value: 0.5
    Name: booster
    Value: gbtree
    Name: csv_weights
    Value: 0
    Name: dsplit
    Value: row
    Name: grow_policy
    Value: depthwise
    Name: lambda_bias
    Value: 0.0
    Name: max_bin
    Value: 256
    Name: max_leaves
    Value: 0
    Name: normalize_type
    Value: tree
    Name: objective
    Value: reg:linear
    Name: one_drop
    Value: 0
    Name: prob_buffer_row
    Value: 1.0
    Name: process_type
    Value: default
    Name: rate_drop
    Value: 0.0
    Name: refresh_leaf
    Value: 1
    Name: sample_type
    Value: uniform
    Name: scale_pos_weight
    Value: 1.0
    Name: silent
    Value: 0
    Name: sketch_eps
    Value: 0.03
    Name: skip_drop
    Value: 0.0
    Name: tree_method

```

```

Value: auto
Name: tweedie_variance_power
Value: 1.5
Stopping Condition:
  Max Runtime In Seconds: 86400
Status:
  Best Training Job:
    Creation Time: 2019-10-17T01:16:14Z
    Final Hyper Parameter Tuning Job Objective Metric:
      Metric Name: validation:error
      Value:
    Objective Status: Succeeded
    Training End Time: 2019-10-17T01:20:24Z
    Training Job Arn: arn:aws:sagemaker:us-east-2:123456789012:training-job/
xgboostha92f5e3cf07b11e9bf6c06d6-009-4sample
    Training Job Name: xgboostha92f5e3cf07b11e9bf6c06d6-009-4c7a3059
    Training Job Status: Completed
    Training Start Time: 2019-10-17T01:18:35Z
    Tuned Hyper Parameters:
      Name: num_round
      Value: 18
    Hyper Parameter Tuning Job Status: Completed
    Last Check Time: 2019-10-17T01:21:01Z
    Sage Maker Hyper Parameter Tuning Job Name: xgboostha92f5e3cf07b11e9bf6c06d6
    Training Job Status Counters:
      Completed: 10
      In Progress: 0
      Non Retryable Error: 0
      Retryable Error: 0
      Stopped: 0
      Total Error: 0
    Events: <none>

```

## [View Logs from Hyperparameter Tuning Jobs](#)

Hyperparameter tuning jobs do not have logs, but all training jobs launched by them do have logs. These logs can be accessed as if they were a normal training job. For more information, see [View Logs from Training Jobs](#).

## [Delete a Hyperparameter tuning Job](#)

Use the following command to stop a hyperparameter job in SageMaker.

```
kubectl delete hyperparametertuningjob xgboost-mnist-hpo
```

This command removes the hyperparameter tuning job and associated training jobs from your Kubernetes cluster and stops them in SageMaker. Jobs that have stopped or completed do not incur any charges for SageMaker resources. SageMaker does not delete hyperparameter tuning jobs. Stopped jobs continue to show on the SageMaker Console.

Your output should look like the following:

```
hyperparametertuningjob.sagemaker.aws.amazon.com "xgboost-mnist-hpo" deleted
```

**Note:** The delete command takes about 2 minutes to clean up the resources from SageMaker.

## [BatchTransformJobs operator](#)

Batch transform job operators reconcile your specified batch transform job spec to SageMaker by launching it in SageMaker. You can learn more about SageMaker batch transform job in the SageMaker [CreateTransformJob API documentation](#).

## Create a BatchTransformJob Using a Simple YAML File

1. Download the sample YAML file for the batch transform job using the following command:

```
wget https://raw.githubusercontent.com/aws/amazon-sagemaker-operator-for-k8s/master/samples/xgboost-mnist-batchtransform.yaml
```

2. Edit the file `xgboost-mnist-batchtransform.yaml` to change necessary parameters to replace the `inputdataconfig` with your input data and `s3OutputPath` with your Amazon S3 buckets that the SageMaker execution role has write access to.
3. Apply the YAML file using the following command:

```
kubectl apply -f xgboost-mnist-batchtransform.yaml
```

## Create a BatchTransformJob Using a Helm Chart

You can use Helm Charts to run batch transform jobs.

### Get the Helm installer directory

Clone the GitHub repo to get the source using the following command:

```
git clone https://github.com/aws/amazon-sagemaker-operator-for-k8s.git
```

### Configure the Helm Chart

Navigate to the `amazon-sagemaker-operator-for-k8s/hack/charts/batch-transform-jobs/` folder.

Edit the `values.yaml` file to replace the `inputdataconfig` with your input data and `outputPath` with your S3 buckets to which the SageMaker execution role has write access.

### Create a Batch Transform Job

1. Use the following command to create a batch transform job:

```
helm install . --generate-name
```

Your output should look like the following:

```
NAME: chart-1574292948
LAST DEPLOYED: Wed Nov 20 23:35:49 2019
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thanks for installing the sagemaker-k8s-batch-transform-job.
```

2. To verify that the Helm Chart was created successfully, run the following command:

```
helm ls
NAME          STATUS        LAST DEPLOYED   REVISION    UPDATED          APP VERSION
chart-1474292948     deployed      2019-11-20 23:35:49.9136092 +0000 UTC
```

```
chart-1474292948      default      1          2019-11-20 23:35:49.9136092
+0000 UTC  deployed    sagemaker-k8s-hyperparametertuningjob-0.1.0
chart-1574292948      default      1          2019-11-20 23:35:49.9136092
+0000 UTC  deployed    sagemaker-k8s-trainingjob-0.1.0
rolebased-1574291698  default      1          2019-11-20 23:14:59.6777082
+0000 UTC  deployed    sagemaker-k8s-operator-0.1.0
```

This command creates a `BatchTransformJob` Kubernetes resource. The operator launches the actual transform job in SageMaker and updates the `BatchTransformJob` Kubernetes resource to reflect the status of the job in SageMaker. You incur charges for SageMaker resources used during the duration of your job. You do not incur any charges once your job completes or stops.

**Note:** SageMaker does not allow you to update a running batch transform job. You cannot edit any parameter and re-apply the file/config. You must either change the metadata name or delete the existing job and create a new one. Similar to existing training job operators like `TFJob` in Kubeflow, update is not supported.

### List Batch Transform Jobs

Use the following command to list all jobs created using the Kubernetes operator:

```
kubectl get batchtransformjob
```

Your output should look like the following:

NAME	STATUS	CREATION-TIME	SAGEMAKER-JOB-NAME
xgboost-mnist-batch-transform a88fb19809b511eaac440aa8axgboost	Completed	2019-11-18T03:44:00Z	xgboost-mnist-

A batch transform job will continue to be listed after the job has completed or failed. You can remove a `hyperparametertuningjob` from the list by following the Delete a Batch Transform Job steps. Jobs that have completed or stopped do not incur any charges for SageMaker resources.

### Batch Transform Status Values

The `STATUS` field can be one of the following values:

- `Completed`
- `InProgress`
- `Failed`
- `Stopped`
- `Stopping`

These statuses come directly from the SageMaker official [API documentation](#).

In addition to the official SageMaker status, it is possible for `STATUS` to be `SynchronizingK8sJobWithSageMaker`. This means that the operator has not yet processed the job.

### Describe a Batch Transform Job

You can obtain debugging details using the `describe` `kubectl` verb by running the following command.

```
kubectl describe batchtransformjob xgboost-mnist-batch-transform
```

Your output should look like the following:

```

Name:          xgboost-mnist-batch-transform
Namespace:     default
Labels:        <none>
Annotations:   kubectl.kubernetes.io/last-applied-configuration:
                {"apiVersion":"sagemaker.aws.amazon.com/v1",
                 "kind":"BatchTransformJob", "metadata": {"annotations": {}, "name": "xgboost-mnist", "namespace": ...}}
API Version:  sagemaker.aws.amazon.com/v1
Kind:          BatchTransformJob
Metadata:
  Creation Timestamp: 2019-11-18T03:44:00Z
  Finalizers:
    sagemaker-operator-finalizer
  Generation: 2
  Resource Version: 21990924
  Self Link: /apis/sagemaker.aws.amazon.com/v1/namespaces/default/batchtransformjobs/xgboost-mnist
  UID: a88fb198-09b5-11ea-ac44-0aa8a9UIDNUM
Spec:
  Model Name: TrainingJob-20190814SMJOB-IKEB
  Region: us-east-1
  Transform Input:
    Content Type: text/csv
    Data Source:
      S 3 Data Source:
        S 3 Data Type: S3Prefix
        S 3 Uri: s3://my-bucket/mnist_kmeans_example/input
  Transform Job Name: xgboost-mnist-a88fb19809b511eaac440aa8a9SMJOB
  Transform Output:
    S 3 Output Path: s3://my-bucket/mnist_kmeans_example/output
  Transform Resources:
    Instance Count: 1
    Instance Type: ml.m4.xlarge
Status:
  Last Check Time: 2019-11-19T22:50:40Z
  Sage Maker Transform Job Name: xgboost-mnist-a88fb19809b511eaac440aa8a9SMJOB
  Transform Job Status: Completed
Events: <none>

```

## View Logs from Batch Transform Jobs

Use the following command to see the logs from the `xgboost-mnist` batch transform job:

```
kubectl smlogs batchtransformjob xgboost-mnist-batch-transform
```

## Delete a Batch Transform Job

Use the following command to stop a batch transform job in SageMaker.

```
kubectl delete batchTransformJob xgboost-mnist-batch-transform
```

Your output should look like the following:

```
batchtransformjob.sagemaker.aws.amazon.com "xgboost-mnist" deleted
```

This command removes the batch transform job from your Kubernetes cluster, as well as stops them in SageMaker. Jobs that have stopped or completed do not incur any charges for SageMaker resources. Delete takes about 2 minutes to clean up the resources from SageMaker.

**Note:** SageMaker does not delete batch transform jobs. Stopped jobs continue to show on the SageMaker console.

## Real-time inference

HostingDeployments support creating and deleting an endpoint, as well as updating an existing endpoint. The hosting deployment operator reconciles your specified hosting deployment job spec to SageMaker by creating models, endpoint-configs and endpoints in SageMaker. You can learn more about SageMaker inference in the SageMaker [CreateEndpoint API documentaiton](#).

### Configure a HostingDeployment Resource

Download the sample YAML file for the hosting deployment job using the following command:

```
wget https://raw.githubusercontent.com/aws/amazon-sagemaker-operator-for-k8s/master/samples/xgboost-mnist-hostingdeployment.yaml
```

The `xgboost-mnist-hostingdeployment.yaml` file has the following components that can be edited as required:

- *ProductionVariants*. A production variant is a set of instances serving a single model. SageMaker load-balances between all production variants according to set weights.
- *Models*. A model is the containers and execution role ARN necessary to serve a model. It requires at least a single container.
- *Containers*. A container specifies the dataset and serving image. If you are using your own custom algorithm instead of an algorithm provided by SageMaker, the inference code must meet SageMaker requirements. For more information, see [Using Your Own Algorithms with SageMaker](#).

### Create a HostingDeployment

To create a HostingDeployment, use `kubectl` to apply the file `hosting.yaml` with the following command:

```
kubectl apply -f hosting.yaml
```

SageMaker creates an endpoint with the specified configuration. You incur charges for SageMaker resources used during the lifetime of your endpoint. You do not incur any charges once your endpoint is deleted.

The creation process takes approximately 10 minutes.

### List HostingDeployments

To verify that the HostingDeployment was created, use the following command:

```
kubectl get hostingdeployments
```

Your output should look like the following:

NAME	STATUS	SAGEMAKER-ENDPOINT-NAME
host-xgboost	Creating	host-xgboost-def0e83e0d5f11eaaa450aSMLOGS

### HostingDeployment Status Values

The status field can be one of several values:

- **SynchronizingK8sJobWithSageMaker**: The operator is preparing to create the endpoint.
- **ReconcilingEndpoint**: The operator is creating, updating, or deleting endpoint resources. If the HostingDeployment remains in this state, use `kubectl describe` to see the reason in the Additional field.

- **OutOfService**: Endpoint is not available to take incoming requests.
- **Creating**: [CreateEndpoint](#) is executing.
- **Updating**: [UpdateEndpoint](#) or [UpdateEndpointWeightsAndCapacities](#) is executing.
- **SystemUpdating**: Endpoint is undergoing maintenance and cannot be updated or deleted or re-scaled until it has completed. This maintenance operation does not change any customer-specified values such as VPC config, KMS encryption, model, instance type, or instance count.
- **RollingBack**: Endpoint fails to scale up or down or change its variant weight and is in the process of rolling back to its previous configuration. Once the rollback completes, endpoint returns to an **InService** status. This transitional status only applies to an endpoint that has autoscaling enabled and is undergoing variant weight or capacity changes as part of an [UpdateEndpointWeightsAndCapacities](#) call or when the [UpdateEndpointWeightsAndCapacities](#) operation is called explicitly.
- **InService**: Endpoint is available to process incoming requests.
- **Deleting**: [DeleteEndpoint](#) is executing.
- **Failed**: Endpoint could not be created, updated, or re-scaled. Use [DescribeEndpoint:FailureReason](#) for information about the failure. [DeleteEndpoint](#) is the only operation that can be performed on a failed endpoint.

### Describe a HostingDeployment

You can obtain debugging details using the `describe kubectl` verb by running the following command.

```
kubectl describe hostingdeployment
```

Your output should look like the following:

```
Name:          host-xgboost
Namespace:    default
Labels:        <none>
Annotations:   kubectl.kubernetes.io/last-applied-configuration:
               {"apiVersion":"sagemaker.aws.amazon.com/v1",
                "kind":"HostingDeployment",
                "metadata": {"annotations": {}, "name": "host-xgboost", "namespace": "def..."}
               }
API Version:  sagemaker.aws.amazon.com/v1
Kind:         HostingDeployment
Metadata:
  Creation Timestamp:  2019-11-22T19:40:00Z
  Finalizers:
    sagemaker-operator-finalizer
  Generation:      1
  Resource Version: 4258134
  Self Link:       /apis/sagemaker.aws.amazon.com/v1/namespaces/default/hostingdeployments/host-xgboost
  UID:             def0e83e-0d5f-11ea-aa45-0a3507uiduid
Spec:
  Containers:
    Container Hostname:  xgboost
    Image:           123456789012.dkr.ecr.us-east-2.amazonaws.com/xgboost:latest
    Model Data URL:  s3://my-bucket/inference/xgboost-mnist/model.tar.gz
  Models:
    Containers:
      xgboost
    Execution Role Arn:  arn:aws:iam::123456789012:role/service-role/AmazonSageMaker-ExecutionRole
    Name:             xgboost-model
    Primary Container: xgboost
  Production Variants:
    Initial Instance Count:  1
    Instance Type:        ml.c5.large
```

```

Model Name:           xgboost-model
Variant Name:        all-traffic
Region:              us-east-2
Status:
  Creation Time:    2019-11-22T19:40:04Z
  Endpoint Arn:     arn:aws:sagemaker:us-east-2:123456789012:endpoint/host-xgboost-
def0e83e0d5f11eaaaexample
  Endpoint Config Name: host-xgboost-1-def0e83e0d5f11e-e08f6c510d5f11eaaa450aexample
  Endpoint Name:      host-xgboost-def0e83e0d5f11eaaa450a350733ba06
  Endpoint Status:   Creating
  Endpoint URL:     https://runtime.sagemaker.us-east-2.amazonaws.com/endpoints/host-
xgboost-def0e83e0d5f11eaaaexample/invocations
  Last Check Time:   2019-11-22T19:43:57Z
  Last Modified Time: 2019-11-22T19:40:04Z
  Model Names:
    Name: xgboost-model
    Value: xgboost-model-1-def0e83e0d5f11-df5cc9fd0d5f11eaaa450aexample
Events: <none>

```

The status field provides more information using the following fields:

- **Additional:** Additional information about the status of the hosting deployment. This field is optional and only gets populated in case of error.
- **Creation Time:** When the endpoint was created in SageMaker.
- **Endpoint ARN:** The SageMaker endpoint ARN.
- **Endpoint Config Name:** The SageMaker name of the endpoint configuration.
- **Endpoint Name:** The SageMaker name of the endpoint.
- **Endpoint Status:** The status of the endpoint.
- **Endpoint URL:** The HTTPS URL that can be used to access the endpoint. For more information, see [Deploy a Model on SageMaker Hosting Services](#).
- **FailureReason:** If a create, update, or delete command fails, the cause is shown here.
- **Last Check Time:** The last time the operator checked the status of the endpoint.
- **Last Modified Time:** The last time the endpoint was modified.
- **Model Names:** A key-value pair of HostingDeployment model names to SageMaker model names.

## Invoking the Endpoint

Once the endpoint status is `InService`, you can invoke the endpoint in two ways: using the Amazon CLI, which does authentication and URL request signing, or using an HTTP client like cURL. If you use your own client, you need to do Amazonv4 URL signing and authentication on your own.

To invoke the endpoint using the Amazon CLI, run the following command. Make sure to replace the Region and endpoint-name with your endpoint's Region and SageMaker endpoint name. This information can be obtained from the output of `kubectl describe`.

```
# Invoke the endpoint with mock input data.
aws sagemaker-runtime invoke-endpoint \
--region us-east-2 \
--endpoint-name <endpoint name> \
--body $(seq 784 | xargs echo | sed 's/ //g') \
>(cat) \
--content-type text/csv > /dev/null
```

For example, if your Region is `us-east-2` and your endpoint config name is `host-xgboost-f56b6b280d7511ea824b129926example`, then the following command would invoke the endpoint:

```
aws sagemaker-runtime invoke-endpoint \
--region us-east-2 \
--endpoint-name host-xgboost-f56b6b280d7511ea824b1299example \
--body $(seq 784 | xargs echo | sed 's/ //g') \
```

```
>(cat) \
--content-type text/csv > /dev/null
4.95847082138
```

Here, 4.95847082138 is the prediction from the model for the mock data.

## Update HostingDeployment

- Once a HostingDeployment has a status of `InService`, it can be updated. It might take about 10 minutes for HostingDeployment to be in service. To verify that the status is `InService`, use the following command:

```
kubectl get hostingdeployments
```

- The HostingDeployment can be updated before the status is `InService`. The operator waits until the SageMaker endpoint is `InService` before applying the update.

To apply an update, modify the `hosting.yaml` file. For example, change the `initialInstanceCount` field from 1 to 2 as follows:

```
apiVersion: sagemaker.aws.amazon.com/v1
kind: HostingDeployment
metadata:
  name: host-xgboost
spec:
  region: us-east-2
  productionVariants:
    - variantName: all-traffic
      modelName: xgboost-model
      initialInstanceCount: 2
      instanceType: ml.c5.large
  models:
    - name: xgboost-model
      executionRoleArn: arn:aws:iam::123456789012:role/service-role/
        AmazonSageMaker-ExecutionRole
      primaryContainer: xgboost
      containers:
        - xgboost
  containers:
    - containerHostname: xgboost
      modelDataUrl: s3://my-bucket/inference/xgboost-mnist/model.tar.gz
      image: 123456789012.dkr.ecr.us-east-2.amazonaws.com/xgboost:latest
```

- Save the file, then use `kubectl` to apply your update as follows. You should see the status change from `InService` to `ReconcilingEndpoint`, then `Updating`.

```
$ kubectl apply -f hosting.yaml
hostingdeployment.sagemaker.aws.amazon.com/host-xgboost configured

$ kubectl get hostingdeployments
NAME           STATUS          SAGEMAKER-ENDPOINT-NAME
host-xgboost   ReconcilingEndpoint  host-xgboost-def0e83e0d5f11eaaa450a350abcdef

$ kubectl get hostingdeployments
NAME           STATUS          SAGEMAKER-ENDPOINT-NAME
host-xgboost   Updating       host-xgboost-def0e83e0d5f11eaaa450a3507abcdef
```

SageMaker deploys a new set of instances with your models, switches traffic to use the new instances, and drains the old instances. As soon as this process begins, the status becomes `Updating`. After the update is complete, your endpoint becomes `InService`. This process takes approximately 10 minutes.

## Delete the HostingDeployment

1. Use `kubectl` to delete a HostingDeployment with the following command:

```
kubectl delete hostingdeployments host-xgboost
```

Your output should look like the following:

```
hostingdeployment.sagemaker.aws.amazon.com "host-xgboost" deleted
```

2. To verify that the hosting deployment has been deleted, use the following command:

```
kubectl get hostingdeployments  
No resources found.
```

Endpoints that have been deleted do not incur any charges for SageMaker resources.

# SageMaker Components for Kubeflow Pipelines

This document outlines how to use SageMaker Components for Kubeflow Pipelines (KFP). With these pipeline components, you can create and monitor training, tuning, endpoint deployment, and batch transform jobs in SageMaker. By running Kubeflow Pipeline jobs on SageMaker, you move data processing and training jobs from the Kubernetes cluster to SageMaker's machine learning-optimized managed service. This document assumes prior knowledge of Kubernetes and Kubeflow.

## Contents

- [What is Kubeflow Pipelines? \(p. 2294\)](#)
- [Kubeflow Pipeline components \(p. 2295\)](#)
- [IAM permissions \(p. 2296\)](#)
- [Converting Pipelines to use SageMaker \(p. 2296\)](#)
- [Using SageMaker Components \(p. 2297\)](#)

## What is Kubeflow Pipelines?

Kubeflow Pipelines (KFP) is a platform for building and deploying portable, scalable machine learning (ML) workflows based on Docker containers. The Kubeflow Pipelines platform consists of the following:

- A user interface (UI) for managing and tracking experiments, jobs, and runs.
- An engine (Argo) for scheduling multi-step ML workflows.
- A Python SDK for defining and manipulating pipelines and components.
- Notebooks for interacting with the system using the SDK.

A pipeline is a description of an ML workflow expressed as a directed acyclic [graph](#) as shown in the following diagram. Every step in the workflow is expressed as a Kubeflow Pipeline [component](#), which is a Python module.

If your data has been preprocessed, the standard pipeline takes a subset of the data and runs hyperparameter optimization of the model. The pipeline then trains a model with the full dataset using the optimal hyperparameters. This model is used for both batch inference and endpoint creation.

For more information on Kubeflow Pipelines, see the [Kubeflow Pipelines documentation](#).

## Kubeflow Pipeline components

A Kubeflow Pipeline component is a set of code used to execute one step in a Kubeflow pipeline. Components are represented by a Python module that is converted into a Docker image. These components make it fast and easy to write pipelines for experimentation and production environments without having to interact with the underlying Kubernetes infrastructure.

### What do SageMaker Components for Kubeflow Pipelines provide?

SageMaker Components for Kubeflow Pipelines offer an alternative to launching compute-intensive jobs in SageMaker. These components integrate SageMaker with the portability and orchestration of Kubeflow Pipelines. Using the SageMaker components, each of the jobs in the pipeline workflow runs on SageMaker instead of the local Kubernetes cluster. The job parameters, status, logs, and outputs from SageMaker are still accessible from the Kubeflow Pipelines UI. The following SageMaker components have been created to integrate six key SageMaker features into your ML workflows. You can create a Kubeflow Pipeline built entirely using these components, or integrate individual components into your workflow as needed.

There is no additional charge for using SageMaker Components for Kubeflow Pipelines. You incur charges for any SageMaker resources you use through these components.

#### [Training components](#)

##### **Training**

The Training component allows you to submit SageMaker Training jobs directly from a Kubeflow Pipelines workflow. For more information, see [SageMaker Training Kubeflow Pipelines component](#).

##### **Hyperparameter Optimization**

The Hyperparameter Optimization component enables you to submit hyperparameter tuning jobs to SageMaker directly from a Kubeflow Pipelines workflow. For more information, see [SageMaker hyperparameter optimization Kubeflow Pipeline component](#).

##### **Processing**

The Processing component enables you to submit processing jobs to SageMaker directly from a Kubeflow Pipelines workflow. For more information, see [SageMaker Processing Kubeflow Pipeline component](#).

#### [Inference components](#)

##### **Hosting Deploy**

The Deploy component enables you to deploy a model in SageMaker Hosting from a Kubeflow Pipelines workflow. For more information, see [SageMaker Hosting Services - Create Endpoint Kubeflow Pipeline component](#).

##### **Batch Transform component**

The Batch Transform component enables you to run inference jobs for an entire dataset in SageMaker from a Kubeflow Pipelines workflow. For more information, see [SageMaker Batch Transform Kubeflow Pipeline component](#).

#### [Ground Truth components](#)

**Ground Truth** The Ground Truth component enables you to submit SageMaker Ground Truth labeling jobs directly from a Kubeflow Pipelines workflow. For more information, see [SageMaker Ground Truth Kubeflow Pipelines component](#).

## Workteam

The Workteam component enables you to create SageMaker private workteam jobs directly from a Kubeflow Pipelines workflow. For more information, see [SageMaker create private workteam Kubeflow Pipelines component](#).

## IAM permissions

Deploying Kubeflow Pipelines with SageMaker components requires the following three levels of IAM permissions:

- An IAM user/role to access your Amazon account (**your\_credentials**). Note: You don't need this at all if you already have access to KFP web UI and have your input data in Amazon S3, or if you already have an Amazon Elastic Kubernetes Service (Amazon EKS) cluster with KFP.

You use this user/role from your gateway node, which can be your local machine or a remote instance, to:

- Create an Amazon EKS cluster and install KFP
- Create IAM roles/users
- Create Amazon S3 buckets for your sample input data

The IAM user/role needs the following permissions:

- CloudWatchLogsFullAccess
- [AWSCloudFormationFullAccess](#)
- IAMFullAccess
- AmazonS3FullAccess
- AmazonEC2FullAccess
- AmazonEKSAdminPolicy (Create this policy using the schema from [Amazon EKS Identity-Based Policy Examples](#))

- An IAM role used by KFP pods to access SageMaker (**kfp-example-pod-role**) The KFP pods use this permission to create SageMaker jobs from KFP components. Note: If you want to limit permissions to the KFP pods, create your own custom policy and attach it.

The role needs the following permission:

- AmazonSageMakerFullAccess
- An IAM role used by SageMaker jobs to access resources such as Amazon S3 and Amazon ECR etc. (**kfp-example-sagemaker-execution-role**).

Your SageMaker jobs use this role to:

- Access SageMaker resources
- Input Data from Amazon S3
- Store your output model to Amazon S3

The role needs the following permissions:

- AmazonSageMakerFullAccess
- AmazonS3FullAccess

These are all the IAM users/roles you need to run KFP components for SageMaker.

When you have run the components and have created the SageMaker endpoint, you also need a role with the `sagemaker:InvokeEndpoint` permission to query inference endpoints.

## Converting Pipelines to use SageMaker

You can convert an existing pipeline to use SageMaker by porting your generic Python [processing containers](#) and [training containers](#). If you are using SageMaker for inference, you also need to attach IAM permissions to your cluster and convert an artifact to a model.

## Using SageMaker Components

In this tutorial, you run a pipeline using SageMaker Components for Kubeflow Pipelines to train a classification model using Kmeans with the MNIST dataset. This workflow uses Kubeflow pipelines as the orchestrator and SageMaker as the backend to run the steps in the workflow. For the full code for this and other pipeline examples, see the [Sample SageMaker Kubeflow Pipelines](#). For information on the components used, see the [KubeFlow Pipelines GitHub repository](#).

### Contents

- [Setup \(p. 2297\)](#)
- [Running the Kubeflow Pipeline \(p. 2303\)](#)

## Setup

To use Kubeflow Pipelines (KFP), you need an Amazon Elastic Kubernetes Service (Amazon EKS) cluster and a gateway node to interact with that cluster. The following sections show the steps needed to set up these resources.

### Topics

- [Set up a gateway node \(p. 2297\)](#)
- [Set up an Amazon EKS cluster \(p. 2298\)](#)
- [Install Kubeflow Pipelines \(p. 2298\)](#)
- [Access the KFP UI \(p. 2299\)](#)
- [Create IAM Users/Roles for KFP pods and the SageMaker service \(p. 2300\)](#)
- [Add access to additional IAM users or roles \(p. 2302\)](#)

### Set up a gateway node

A gateway node is used to create an Amazon EKS cluster and access the Kubeflow Pipelines UI. Use your local machine or an Amazon EC2 instance as your gateway node. If you want to use a new Amazon EC2 instance, create one with the latest Ubuntu 18.04 DLAMI version from the Amazon console using the steps in [Launching and Configuring a DLAMI](#).

Complete the following steps to set up your gateway node. Depending on your environment, you may have certain requirements already configured.

1. If you don't have an existing Amazon EKS cluster, create a user named `your_credentials` using the steps in [Creating an IAM User in Your Amazon Account](#). If you have an existing Amazon EKS cluster, use the credentials of the IAM role or user that has access to it.
2. Add the following permissions to your user using the steps in [Changing Permissions for an IAM User](#):
  - `CloudWatchLogsFullAccess`
  - `AWSCloudFormationFullAccess`
  - `IAMFullAccess`
  - `AmazonS3FullAccess`
  - `AmazonEC2FullAccess`
  - `AmazonEKSAdminPolicy` (Create this policy using the schema from [Amazon EKS Identity-Based Policy Examples](#))
3. Install the following on your gateway node to access the Amazon EKS cluster and KFP UI.
  - [Amazon CLI](#). If you are using an IAM user, configure your `Access Key ID`, `Secret Access Key` and preferred Amazon Region by running: `aws configure`
  - [aws-iam-authenticator](#) version 0.1.31 and above
  - [eksctl](#) version above 0.15.

- [kubectl](#) (The version needs to match your Kubernetes version within one minor version)
4. Install boto3.

```
pip install boto3
```

## Set up an Amazon EKS cluster

Run the following steps from the command line of your gateway node to set up an Amazon EKS cluster:

1. If you do not have an existing Amazon EKS cluster, complete the following substeps. If you already have an Amazon EKS cluster, skip this step.
  - a. Run the following from your command line to create an Amazon EKS Cluster with version 1.14 or above. Replace <your-cluster-name> with any name for your cluster.

```
eksctl create cluster --name <your-cluster-name> --region us-east-1 --auto-kubeconfig --timeout=50m --managed --nodes=1
```

- b. When cluster creation is complete, verify that you have access to the cluster using the following command.

```
kubectl get nodes
```

2. Verify that the current kubectl context is the cluster you want to use with the following command. The current context is marked with an asterisk (\*) in the output.

```
kubectl config get-contexts
```

CURRENT NAME	CLUSTER
*	<username>@<clusternode>.us-east-1.eksctl.io <clusternode>.us-east-1.eksctl.io

3. If the desired cluster is not configured as your current default, update the default with the following command.

```
aws eks update-kubeconfig --name <clusternode> --region us-east-1
```

## Install Kubeflow Pipelines

Run the following steps from the command line of your gateway node to install Kubeflow Pipelines on your cluster.

1. Install Kubeflow Pipelines on your cluster by following step 1 of [Deploying Kubeflow Pipelines documentation](#). Your KFP version must be 0.5.0 or above.
2. Verify that the Kubeflow Pipelines service and other related resources are running.

```
kubectl -n kubeflow get all | grep pipeline
```

Your output should look like the following.

pod/ml-pipeline-6b88c67994-kdtjv 2d	1/1	Running	0
pod/ml-pipeline-persistenceagent-64d74dfdbf-66stk 2d	1/1	Running	0
pod/ml-pipeline-scheduledworkflow-65bdf46db7-5x9qj 2d	1/1	Running	0

pod/ml-pipeline-ui-66cc4cffb6-cmsdb	1/1	Running	0
2d			
pod/ml-pipeline-viewer-crd-6db65ccc4-wqlzj	1/1	Running	0
2d			
pod/ml-pipeline-visualizationserver-9c47576f4-bqmx4	1/1	Running	0
2d			
service/ml-pipeline	ClusterIP	10.100.170.170	<none>
8888/TCP,8887/TCP	2d		
service/ml-pipeline-ui	ClusterIP	10.100.38.71	<none>
80/TCP	2d		
service/ml-pipeline-visualizationserver	ClusterIP	10.100.61.47	<none>
8888/TCP	2d		
deployment.apps/ml-pipeline	1/1	1	1
deployment.apps/ml-pipeline-persistenceagent	1/1	1	1
deployment.apps/ml-pipeline-scheduledworkflow	1/1	1	1
deployment.apps/ml-pipeline-ui	1/1	1	1
deployment.apps/ml-pipeline-viewer-crd	1/1	1	1
deployment.apps/ml-pipeline-visualizationserver	1/1	1	1
replicaset.apps/ml-pipeline-6b88c67994		1	1
2d			
replicaset.apps/ml-pipeline-persistenceagent-64d74dfdbf		1	1
2d			
replicaset.apps/ml-pipeline-scheduledworkflow-65bdf46db7		1	1
2d			
replicaset.apps/ml-pipeline-ui-66cc4cffb6		1	1
2d			
replicaset.apps/ml-pipeline-viewer-crd-6db65ccc4		1	1
2d			
replicaset.apps/ml-pipeline-visualizationserver-9c47576f4		1	1
2d			

## Access the KFP UI

The Kubeflow Pipelines UI is used for managing and tracking experiments, jobs, and runs on your cluster. You can use port forwarding to access the Kubeflow Pipelines UI from your gateway node.

### Set up port forwarding to the KFP UI service

Run the following from the command line of your gateway node:

- Verify that the KFP UI service is running using the following command:

```
kubectl -n kubeflow get service ml-pipeline-ui

NAME           TYPE      CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
ml-pipeline-ui  ClusterIP  10.100.38.71  <none>        80/TCP     2d22h
```

- Run the following command to set up port forwarding to the KFP UI service. This forwards the KFP UI to port 8080 on your gateway node and allows you to access the KFP UI from your browser.

```
kubectl port-forward -n kubeflow service/ml-pipeline-ui 8080:80
```

The port forward from your remote machine drops if there is no activity. Run this command again if your dashboard is unable to get logs or updates. If the commands return an error, ensure that there is no process already running on the port you are trying to use.

## Access the KFP UI service

Your method of accessing the KFP UI depends on your gateway node type.

- Local machine as the gateway node

1. Access the dashboard in your browser as follows:

```
http://localhost:8080
```

2. Choose **Pipelines** to access the pipelines UI.

- Amazon EC2 instance as the gateway node

1. You need to set up an SSH tunnel on your Amazon EC2 instance to access the Kubeflow dashboard from your local machine's browser.

From a new terminal session in your local machine, run the following. Replace <public-DNS-of-gateway-node> with the IP address of your instance found on the Amazon EC2 console. You can also use the public DNS. Replace <path\_to\_key> with the path to the pem key used to access the gateway node.

```
public_DNS_address=<public-DNS-of-gateway-node>
key=<path_to_key>

on Ubuntu:
ssh -i ${key} -L 9000:localhost:8080 ubuntu@${public_DNS_address}

or on Amazon Linux:
ssh -i ${key} -L 9000:localhost:8080 ec2-user@${public_DNS_address}
```

2. Access the dashboard in your browser.

```
http://localhost:9000
```

3. Choose **Pipelines** to access the KFP UI.

## Create IAM Users/Roles for KFP pods and the SageMaker service

You now have a Kubernetes cluster with Kubeflow set up. To run SageMaker Components for Kubeflow Pipelines, the Kubeflow Pipeline pods need access to SageMaker. In this section, you create IAM users/roles to be used by Kubeflow Pipeline pods and SageMaker.

### Create a KFP execution role

Run the following from the command line of your gateway node:

1. Enable OIDC support on the Amazon EKS cluster with the following command. Replace <cluster\_name> with the name of your cluster and <cluster\_region> with the region your cluster is in.

```
eksctl utils associate-iam-oidc-provider --cluster <cluster-name> \
--region <cluster-region> --approve
```

2. Run the following to get the **OIDC** issuer URL. This URL is in the form `https://oidc.eks.<region>.amazonaws.com/id/<OIDC_ID>`.

```
aws eks describe-cluster --region <cluster-region> --name <cluster-name> --query
"cluster.identity.oidc.issuer" --output text
```

3. Run the following to create a file named `trust.json`. Replace <OIDC\_URL> with your OIDC issuer URL. Don't include `https://` when in your OIDC issuer URL. Replace <AWS\_account\_number> with your Amazon account number.

```

OIDC_URL="<OIDC-URL>"
AWS_ACC_NUM="<AWS-account-number>"

# Run this to create trust.json file
cat <<EOF > trust.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::${AWS_ACC_NUM}:oidc-provider/${OIDC_URL}"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "${OIDC_URL}:aud": "sts.amazonaws.com",
                    "${OIDC_URL}:sub": "system:serviceaccount:kubeflow:pipeline-runner"
                }
            }
        }
    ]
}
EOF

```

4. Create an IAM role named `kfp-example-pod-role` using `trust.json` using the following command. This role is used by KFP pods to create SageMaker jobs from KFP components. Note the ARN returned in the output.

```

aws iam create-role --role-name kfp-example-pod-role --assume-role-policy-document
file://trust.json
aws iam attach-role-policy --role-name kfp-example-pod-role --policy-arn
arn:aws:iam::aws:policy/AmazonSageMakerFullAccess
aws iam get-role --role-name kfp-example-pod-role --output text --query 'Role.Arn'

```

5. Edit your pipeline-runner service account with the following command.

```
kubectl edit -n kubeflow serviceaccount pipeline-runner
```

6. In the file, add the following Amazon EKS role annotation and replace `<role_arn>` with your role ARN.

```
eks.amazonaws.com/role-arn: <i><role-arn></i>
```

7. Your file should look like the following when you've added the Amazon EKS role annotation. Save the file.

```

apiVersion: v1
kind: ServiceAccount
metadata:
  annotations:
    eks.amazonaws.com/role-arn: <i><role-arn></i>
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","kind":"ServiceAccount","metadata":{"annotations":{},"labels":{"app":"pipeline-runner","app.kubernetes.io/component":"pipelines-runner","app.kubernetes.io/instance":"pipelines-runner-0.2.0","app.kubernetes.io/managed-by":"kfctl","app.kubernetes.io/name":"pipelines-runner","app.kubernetes.io/part-of":"kubeflow","app.kubernetes.io/version":"0.2.0"},"name":"pipeline-runner","namespace":"kubeflow"}}
    creationTimestamp: "2020-04-16T05:48:06Z"
    labels:

```

```

app: pipeline-runner
app.kubernetes.io/component: pipelines-runner
app.kubernetes.io/instance: pipelines-runner-0.2.0
app.kubernetes.io/managed-by: kfctl
app.kubernetes.io/name: pipelines-runner
app.kubernetes.io/part-of: kubeflow
app.kubernetes.io/version: 0.2.0
name: pipeline-runner
namespace: kubeflow
resourceVersion: "11787"
selfLink: /api/v1/namespaces/kubeflow/serviceaccounts/pipeline-runner
uid: d86234bd-7fa5-11ea-a8f2-02934be6dc88
secrets:
- name: pipeline-runner-token-dkjrk

```

## Create an SageMaker execution role

The `kfp-example-sagemaker-execution-role` IAM role is used by SageMaker jobs to access Amazon resources. For more information, see the [IAM Permissions](#) section. You provide this role as an input parameter when running the pipeline.

Run the following to create the role. Note the ARN that is returned in your output.

```

SAGEMAKER_EXECUTION_ROLE_NAME=kfp-example-sagemaker-execution-role

TRUST="{"Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": "*", "Service": "sagemaker.amazonaws.com"}, {"Action": "sts:AssumeRole"} ] }"
aws iam create-role --role-name ${SAGEMAKER_EXECUTION_ROLE_NAME} --assume-role-policy-document "$TRUST"
aws iam attach-role-policy --role-name ${SAGEMAKER_EXECUTION_ROLE_NAME} --policy-arn arn:aws:iam::aws:policy/AmazonSageMakerFullAccess
aws iam attach-role-policy --role-name ${SAGEMAKER_EXECUTION_ROLE_NAME} --policy-arn arn:aws:iam::aws:policy/AmazonS3FullAccess

aws iam get-role --role-name ${SAGEMAKER_EXECUTION_ROLE_NAME} --output text --query 'Role.Arn'

```

## Add access to additional IAM users or roles

If you use an intuitive IDE like Jupyter or want other people in your organization to use the cluster you set up, you can also give them access. The following steps run through this workflow using SageMaker notebooks. An SageMaker notebook instance is a fully managed Amazon EC2 compute instance that runs the Jupyter Notebook App. You use the notebook instance to create and manage Jupyter notebooks to create ML workflows. You can define, compile, deploy, and run your pipeline using the KFP Python SDK or CLI. If you're not using an SageMaker notebook to run Jupyter, you need to install the [Amazon CLI](#) and the latest version of [kubectl](#).

1. Follow the steps in [Create an SageMaker Notebook Instance](#) to create a SageMaker notebook instance if you do not already have one. Give the IAM role for this instance the S3FullAccess permission.
2. Amazon EKS clusters use IAM users and roles to control access to the cluster. The rules are implemented in a config map named `aws-auth`. Only the user/role that has access to the cluster will be able to edit this config map. Run the following from the command line of your gateway node to get the IAM role of the notebook instance you created. Replace `<instance-name>` with the name of your instance.

```

aws sagemaker describe-notebook-instance --notebook-instance-name <instance-name> --region <region> --output text --query 'RoleArn'

```

This command outputs the IAM role ARN in the `arn:aws:iam::<account-id>:role/<role-name>` format. Take note of this ARN.

- Run the following to attach the policies the IAM role. Replace `<role-name>` with the `<role-name>` in your ARN.

```
aws iam attach-role-policy --role-name <role-name> --policy-arn arn:aws:iam::aws:policy/AmazonSageMakerFullAccess
aws iam attach-role-policy --role-name <role-name> --policy-arn arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
aws iam attach-role-policy --role-name <role-name> --policy-arn arn:aws:iam::aws:policy/AmazonS3FullAccess
```

- eksctl provides commands to read and edit the `aws-auth` config map. `system:masters` is one of the default user groups. You add the user to this group. The `system:masters` group has super user permissions to the cluster. You can also create a group with more restrictive permissions or you can bind permissions directly to users. Replace `<IAM-Role-arn>` with the ARN of the IAM role. `<your_username>` can be any unique username.

```
eksctl create iamidentitymapping \
--cluster <cluster-name> \
--arn <IAM-Role-arn> \
--group system:masters \
--username <your-username> \
--region <region>
```

- Open the Jupyter notebook on your SageMaker instance and run the following to verify that it has access to the cluster.

```
aws eks --region <region> update-kubeconfig --name <cluster-name>
kubectl -n kubeflow get all | grep pipeline
```

## Running the Kubeflow Pipeline

Now that setup of your gateway node and Amazon EKS cluster is complete, you can create your classification pipeline. To create your pipeline, you need to define and compile it. You then deploy it and use it to run workflows. You can define your pipeline in Python and use the KFP dashboard, KFP CLI, or Python SDK to compile, deploy, and run your workflows. The full code for the MNIST classification pipeline example is available in the [Kubeflow Github repository](#). To use it, clone the example Python files to your gateway node.

### Topics

- [Prepare datasets \(p. 2303\)](#)
- [Create a Kubeflow Pipeline using SageMaker Components \(p. 2304\)](#)
- [Compile and deploy your pipeline \(p. 2304\)](#)
- [Running predictions \(p. 2306\)](#)
- [View results and logs \(p. 2308\)](#)
- [Cleanup \(p. 2308\)](#)

### Prepare datasets

To run the pipelines, you need to upload the data extraction pre-processing script to an Amazon S3 bucket. This bucket and all resources for this example must be located in the `us-east-1` Amazon Region. If you don't have a bucket, create one using the steps in [Creating a bucket](#).

From the `mnist-kmeans-sagemaker` folder of the Kubeflow repository you cloned on your gateway node, run the following command to upload the `kmeans_preprocessing.py` file to your Amazon S3 bucket. Change `<bucket-name>` to the name of the Amazon S3 bucket you created.

```
aws s3 cp mnist-kmeans-sagemaker/kmeans_preprocessing.py s3://<bucket-name>/mnist_kmeans_example/processing_code/kmeans_preprocessing.py
```

## Create a Kubeflow Pipeline using SageMaker Components

The full code for the MNIST classification pipeline is available in the [Kubeflow Github repository](#). To use it, clone the example Python files to your gateway node.

### Input Parameters

The full MNIST classification pipeline has run-specific parameters for which you must provide values when creating a run. You must provide these parameters for each component of your pipeline. These parameters can also be updated when using other pipelines. We have provided default values for all parameters in the sample classification pipeline file.

The following are the only parameters you need to pass to run the sample pipelines. To pass these parameters, update their entries when creating a new run.

- **Role-ARN:** This must be the ARN of an IAM role that has full SageMaker access in your Amazon account. Use the ARN of `kfp-example-pod-role`.
- **Bucket:** This is the name of the Amazon S3 bucket that you uploaded the `kmeans_preprocessing.py` file to.

You can adjust any of the input parameters using the KFP UI and trigger your run again.

## Compile and deploy your pipeline

After defining the pipeline in Python, you must compile the pipeline to an intermediate representation before you can submit it to the Kubeflow Pipelines service. The intermediate representation is a workflow specification in the form of a YAML file compressed into a tar.gz file. You need the KFP SDK to compile your pipeline.

### Install KFP SDK

Run the following from the command line of your gateway node:

1. Install the KFP SDK following the instructions in the [Kubeflow pipelines documentation](#).
2. Verify that the KFP SDK is installed with the following command:

```
pip show kfp
```

3. Verify that `dsl-compile` has been installed correctly as follows:

```
which dsl-compile
```

## Compile your pipeline

You have three options to interact with Kubeflow Pipelines: KFP UI, KFP CLI, or the KFP SDK. The following sections illustrate the workflow using the KFP UI and CLI.

Complete the following from your gateway node to compile your pipeline.

1. Modify your Python file with your Amazon S3 bucket name and IAM role ARN.
2. Use the `dsl-compile` command from the command line to compile your pipeline as follows. Replace `<path-to-python-file>` with the path to your pipeline and `<path-to-output>` with the location where you want your tar.gz file to be.

```
dsl-compile --py <path-to-python-file> --output <path-to-output>
```

## Upload and run the pipeline using the KFP CLI

Complete the following steps from the command line of your gateway node. KFP organizes runs of your pipeline as experiments. You have the option to specify an experiment name. If you do not specify one, the run will be listed under **Default** experiment.

1. Upload your pipeline as follows:

```
kfp pipeline upload --pipeline-name <pipeline-name> <path-to-output-tar.gz>
```

Your output should look like the following. Take note of the ID.

```
Pipeline 29c3ff21-49f5-4dfe-94f6-618c0e2420fe has been submitted

Pipeline Details
-----
ID      29c3ff21-49f5-4dfe-94f6-618c0e2420fe
Name    sm-pipeline
Description
Uploaded at 2020-04-30T20:22:39+00:00
...
...
```

2. Create a run using the following command. The KFP CLI run command currently does not support specifying input parameters while creating the run. You need to update your parameters in the Python pipeline file before compiling. Replace `<experiment-name>` and `<job-name>` with any names. Replace `<pipeline-id>` with the ID of your submitted pipeline. Replace `<your-role-arn>` with the ARN of `kfp-example-pod-role`. Replace `<your-bucket-name>` with the name of the Amazon S3 bucket you created.

```
kfp run submit --experiment-name <experiment-name> --run-name <job-name> --pipeline-id <pipeline-id> role_arn="<your-role-arn>" bucket_name="<your-bucket-name>"
```

You can also directly submit a run using the compiled pipeline package created as the output of the `dsl-compile` command.

```
kfp run submit --experiment-name <experiment-name> --run-name <job-name> --package-file <path-to-output> role_arn="<your-role-arn>" bucket_name="<your-bucket-name>"
```

Your output should look like the following:

```
Creating experiment aws.
Run 95084a2c-f18d-4b77-a9da-eba00bf01e63 is submitted
+-----+-----+-----+
+   | run id                  | name    | status   | created at
|   |
+-----+-----+-----+
+-----+-----+-----+
```

```
| 95084a2c-f18d-4b77-a9da-eba00bf01e63 | sm-job | 2020-04-30T20:36:41+00:00
| |
+-----+-----+-----+
|
```

3. Navigate to the UI to check the progress of the job.

### Upload and run the pipeline using the KFP UI

1. On the left panel, choose the **Pipelines** tab.
2. In the upper-right corner, choose **+UploadPipeline**.
3. Enter the pipeline name and description.
4. Choose **Upload a file** and enter the path to the tar.gz file you created using the CLI or with the Python SDK.
5. On the left panel, choose the **Pipelines** tab.
6. Find the pipeline you created.
7. Choose **+CreateRun**.
8. Enter your input parameters.
9. Choose **Run**.

### Running predictions

Once your classification pipeline is deployed, you can run classification predictions against the endpoint that was created by the Deploy component. Use the KFP UI to check the output artifacts for `sagemaker-deploy-model-endpoint_name`. Download the .tgz file to extract the endpoint name or check the SageMaker console in the region you used.

### Configure permissions to run predictions

If you want to run predictions from your gateway node, skip this section.

1. To use any other machine to run predictions, assign the `sagemaker:InvokeEndpoint` permission to the IAM role or IAM user used by the client machine. This permission is used to run predictions.
2. On your gateway node, run the following to create a policy file:

```
cat <<EoF > ./sagemaker-invoke.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:InvokeEndpoint"
            ],
            "Resource": "*"
        }
    ]
}
EoF
```

3. Attach the policy to the client node's IAM role or IAM user.
4. If your client machine has an IAM role attached, run the following. Replace `<your-instance-IAM-role>` with the name of the client node's IAM role. Replace `<path-to-sagemaker-invoke-json>` with the path to the policy file you created.

```
aws iam put-role-policy --role-name <your-instance-IAM-role> --policy-name sagemaker-  
invoke-for-worker --policy-document file://<path-to-sagemaker-invoke-json>
```

5. If your client machine has IAM user credentials configured, run the following. Replace <your\_IAM\_user\_name> with the name of the client node's IAM user. Replace <path-to-sagemaker-invoke-json> with the path to the policy file you created.

```
aws iam put-user-policy --user-name <your-IAM-user-name> --policy-name sagemaker-  
invoke-for-worker --policy-document file://<path-to-sagemaker-invoke-json>
```

## Run predictions

1. Create a Python file from your client machine named `mnist-predictions.py` with the following content. Replace the `ENDPOINT_NAME` variable. This script loads the MNIST dataset, then creates a CSV from those digits and sends it to the endpoint for prediction. It then outputs the results.

```
import boto3
import gzip
import io
import json
import numpy
import pickle

ENDPOINT_NAME='<endpoint-name>'
region = boto3.Session().region_name

# S3 bucket where the original mnist data is downloaded and stored
downloaded_data_bucket = f"jumpstart-cache-prod-{region}"
downloaded_data_prefix = "1p-notebooks-datasets/mnist"

# Download the dataset
s3 = boto3.client("s3")
s3.download_file(downloaded_data_bucket, f"{downloaded_data_prefix}/mnist.pkl.gz",
                 "mnist.pkl.gz")

# Load the dataset
with gzip.open('mnist.pkl.gz', 'rb') as f:
    train_set, valid_set, test_set = pickle.load(f, encoding='latin1')

# Simple function to create a csv from our numpy array
def np2csv(arr):
    csv = io.BytesIO()
    numpy.savetxt(csv, arr, delimiter=',', fmt='%g')
    return csv.getvalue().decode().rstrip()

runtime = boto3.Session(region).client('sagemaker-runtime')

payload = np2csv(train_set[0][30:31])

response = runtime.invoke_endpoint(EndpointName=ENDPOINT_NAME,
                                    ContentType='text/csv',
                                    Body=payload)
result = json.loads(response['Body'].read().decode())
print(result)
```

2. Run the Python file as follows:

```
python mnist-predictions.py
```

## [View results and logs](#)

When the pipeline is running, you can choose any component to check execution details, such as inputs and outputs. This lists the names of created resources.

If the KFP request is successfully processed and an SageMaker job is created, the component logs in the KFP UI provide a link to the job created in SageMaker. The CloudWatch logs are also provided if the job is successfully created.

If you run too many pipeline jobs on the same cluster, you may see an error message that indicates you do not have enough pods available. To fix this, log in to your gateway node and delete the pods created by the pipelines you are not using as follows:

```
kubectl get pods -n kubeflow
kubectl delete pods -n kubeflow <name-of-pipeline-pod>
```

## [Cleanup](#)

When you're finished with your pipeline, you need to clean up your resources.

1. From the KFP dashboard, terminate your pipeline runs if they do not exit properly by choosing **Terminate**.
2. If the **Terminate** option doesn't work, log in to your gateway node and manually terminate all the pods created by your pipeline run as follows:

```
kubectl get pods -n kubeflow
kubectl delete pods -n kubeflow <name-of-pipeline-pod>
```

3. Using your Amazon account, log in to the SageMaker service. Manually stop all training, batch transform, and HPO jobs. Delete models, data buckets, and endpoints to avoid incurring any additional costs. Terminating the pipeline runs does not stop the jobs in SageMaker.

# Using Amazon Augmented AI for Human Review

This feature is not available in the China Regions.

When you use AI applications such as Amazon Rekognition, Amazon Textract, or your custom machine learning (ML) models, you can use Amazon Augmented AI to get human review of low-confidence predictions or random prediction samples.

## What is Amazon Augmented AI?

Amazon Augmented AI (Amazon A2I) is a service that brings human review of ML predictions to all developers by removing the heavy lifting associated with building human review systems or managing large numbers of human reviewers.

Many ML applications require humans to review low-confidence predictions to ensure the results are correct. For example, extracting information from scanned mortgage application forms can require human review due to low-quality scans or poor handwriting. Building human review systems can be time-consuming and expensive because it involves implementing complex processes or *workflows*, writing custom software to manage review tasks and results, and managing large groups of reviewers.

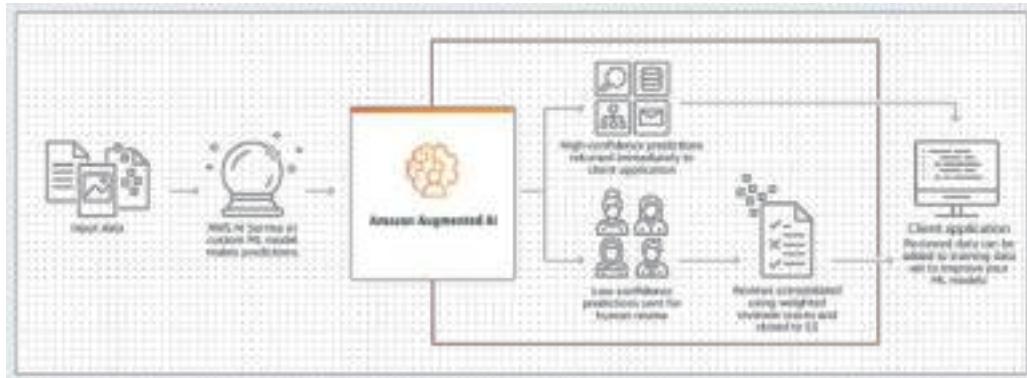
Amazon A2I streamlines building and managing human reviews for ML applications. Amazon A2I provides built-in human review workflows for common ML use cases, such as content moderation and text extraction from documents. You can also create your own workflows for ML models built on SageMaker or any other tools. Using Amazon A2I, you can allow human reviewers to step in when a model is unable to make a high-confidence prediction or to audit its predictions on an ongoing basis.

## Amazon A2I Use Case Examples

The following examples demonstrate how you can use Amazon A2I to integrate a human review loop into your ML application. For each of these examples, you can find a Jupyter Notebook that demonstrates that workflow in [Use Cases and Examples Using Amazon A2I \(p. 2330\)](#).

- **Use Amazon A2I with Amazon Textract** – Have humans review important key-value pairs in single-page documents or have Amazon Textract randomly sample and send documents from your dataset to humans for review.
- **Use Amazon A2I with Amazon Rekognition** – Have humans review unsafe images for explicit adult or violent content if Amazon Rekognition returns a low-confidence score, or have Amazon Rekognition randomly sample and send images from your dataset to humans for review.
- **Use Amazon A2I to review real-time ML inferences** – Use Amazon A2I to review real-time, low-confidence inferences made by a model deployed to a SageMaker hosted endpoint and incrementally train your model using Amazon A2I output data.
- **Use Amazon A2I with Amazon Comprehend** – Have humans review Amazon Comprehend inferences about text data such as sentiment analysis, text syntax, and entity detection.
- **Use Amazon A2I with Amazon Transcribe** – Have humans review Amazon Transcribe transcriptions of video or audio files. Use the results of transcription human review loops to create a custom vocabulary and improve future transcriptions of similar video or audio content.

- **Use Amazon A2I with Amazon Translate** – Have humans review low-confidence translations returned from Amazon Translate.
- **Use Amazon A2I to review tabular data** – Use Amazon A2I to integrate a human review loop into an ML application that uses tabular data.



### Topics

- [Get Started with Amazon Augmented AI \(p. 2310\)](#)
- [Use Cases and Examples Using Amazon A2I \(p. 2330\)](#)
- [Create a Human Review Workflow \(p. 2339\)](#)
- [Delete a Human Review Workflow \(p. 2357\)](#)
- [Create and Start a Human Loop \(p. 2358\)](#)
- [Delete a Human Loop \(p. 2363\)](#)
- [Create and Manage Worker Task Templates \(p. 2366\)](#)
- [Monitor and Manage Your Human Loop \(p. 2376\)](#)
- [Amazon A2I Output Data \(p. 2377\)](#)
- [Permissions and Security in Amazon Augmented AI \(p. 2386\)](#)
- [Use Amazon CloudWatch Events in Amazon Augmented AI \(p. 2392\)](#)
- [Use APIs in Amazon Augmented AI \(p. 2394\)](#)

## Get Started with Amazon Augmented AI

To get started using Amazon Augmented AI, review the [Core Components of Amazon A2I \(p. 2310\)](#) and [Prerequisites to Using Augmented AI \(p. 2314\)](#). Then, use the following documentation to learn how to use the Amazon A2I console and API.

- [Tutorial: Get Started in the Amazon A2I Console \(p. 2314\)](#)
- [Tutorial: Get Started Using the Amazon A2I API \(p. 2320\)](#)

You can also get started using the Amazon A2I API by following a Jupyter Notebook tutorial. See [Use Cases and Examples Using Amazon A2I \(p. 2330\)](#) for a list of notebooks and use cases.

## Core Components of Amazon A2I

Review the following terms to familiarize yourself with the core components of Amazon A2I.

## Task Types

The AI/ML workflow into which you integrate Amazon A2I defines an Amazon A2I *task type*.

Amazon A2I supports:

- Two *built-in task types*: [Amazon Textract key-value pair extraction](#) and [Amazon Rekognition image moderation](#).
- A *custom task type*: Use a custom task type to integrate a human review loop into *any* machine learning workflow. You can use a custom task type to integrate Amazon A2I with other Amazon services like Amazon Comprehend, Amazon Transcribe, and Amazon Translate, as well as your own custom machine learning workflows. To learn more, see [Use Cases and Examples Using Amazon A2I \(p. 2330\)](#).

Select a tab in the following table to see diagrams that illustrate how Amazon A2I works with each task type. Select the task type page using the links in the preceding list to learn more about that task type.

### Amazon Textract – Key-value pair extraction

This image depicts the Amazon A2I built-in workflow with Amazon Textract. On the left, the resources that are required to create an Amazon Textract human review workflow are depicted: an Amazon S3 bucket, activation conditions, a worker task template, and a work team. These resources are used to create a human review workflow, or flow definition. An arrow points right to the next step in the workflow: using Amazon Textract to configure a human loop with the human review workflow. A second arrow points right from this step to the step in which activation conditions specified in the human review workflow are met. This initiates the creation of a human loop. On the right of the image, the human loop is depicted in three steps: 1) the worker UI and tools are generated and the task is made available to workers, 2) workers review input data, and finally, 3) results are saved in Amazon S3.



### Amazon Rekognition – Image moderation

This image depicts the Amazon A2I built-in workflow with Amazon Rekognition. On the left, the resources that are required to create an Amazon Rekognition human review workflow are depicted: an Amazon S3 bucket, activation conditions, a worker task template, and a work team. These resources are used to create a human review workflow, or flow definition. An arrow points right to the next step in the workflow: using Amazon Rekognition to configure a human loop with the human review workflow. A second arrow points right from this step to the step in which activation conditions specified in the human review workflow are met. This initiates the creation of a human

loop. On the right of the image, the human loop is depicted in three steps: 1) the worker UI and tools are generated and the task is made available to workers, 2) workers review input data, and finally, 3) results are saved in Amazon S3.



Custom Task Type

The following image depicts the Amazon A2I custom workflow. A custom ML model is used to generate predictions. The client application filters these predictions using user-defined criteria and determines if a human review is required. If so, these predictions are sent to Amazon A2I for human review. Amazon A2I collects the results of human review in Amazon S3, which can access by the client application. If the filter determines that no human review is needed, predictions can be fed directly to the client application.



## Human Review Workflow (Flow Definition)

You use a human review workflow to specify your human *work team*, to set up your worker UI using a *worker task template*, and to provide information about how workers should complete the review task.

For built-in task types, you also use the human review workflow to identify the conditions under which a human loop is initiated. For example, Amazon Rekognition can perform image content moderation using machine learning. You can use the human review workflow to specify that an image is sent to a human for content moderation review if Amazon Rekognition's confidence is too low.

You can use a human review workflow to create multiple human loops.

You can create a flow definition in the SageMaker console or with the SageMaker API. To learn more about both of these options, see [Create a Human Review Workflow \(p. 2339\)](#).

### Work Team

A *work team* is a group of human workers to whom you send your human review tasks.

When you create a human review workflow, you specify a single work team.

Your work team can come from the [Amazon Mechanical Turk workforce](#), a [vendor-managed workforce](#), or your own [private workforce](#). When you use the private workforce, you can create multiple work teams. Each work team can be used in multiple human review workflows. To learn how to create a workforce and work teams, see [Create and Manage Workforces \(p. 456\)](#).

### Worker Task Template and Human Task UI

You use a *worker task template* to create a worker UI (a *human task UI*) for your human review tasks.

The human task UI displays your input data, such as documents or images, and instructions to workers. It also provides interactive tools that the worker uses to complete your tasks.

For built-in task types, you must use the Amazon A2I worker task template provided for that task type.

## Human Loops

A *human loop* is used to create a single human review job. For each human review job, you can choose the number of workers that are sent a *task* to review a single data object. For example, if you set the number of workers per object to 3 for an image classification labeling job, three workers classify each input image. Increasing the number of workers per object can improve label accuracy.

A human loop is created using a human review workflow as follows:

- For built-in task types, the conditions specified in the human review workflow determine when the human loop is created.
- Human review tasks are sent to the work team specified in the human review workflow.
- The worker task template specified in the human review workflow is used to render the human task UI.

### When do human loops get created?

When you use one of the *built-in task types*, the corresponding Amazon service creates and starts a human loop on your behalf when the conditions specified in your human review workflow are met. For example:

- When you use Augmented AI with Amazon Textract, you can integrate Amazon A2I into a document review task using the API operation `AnalyzeDocument`. A human loop is created every time Amazon Textract returns inferences about key-value pairs that meet the conditions you specify in your human review workflow.
- When you use Augmented AI with Amazon Rekognition, you can integrate Amazon A2I into an image moderation task using the API operation `DetectModerationLabels`. A human loop is created every time Amazon Rekognition returns inferences about image content that meet the conditions you specify in your human review workflow.

When using a *custom task type*, you start a human loop using the [Amazon Augmented AI Runtime API](#). When you call `StartHumanLoop` in your custom application, a task is sent to human reviewers.

To learn how to create and start a human loop, see [Create and Start a Human Loop \(p. 2358\)](#).

To generate these resources and create a human review workflow, Amazon A2I integrates multiple APIs, including the Amazon Augmented AI Runtime Model, the SageMaker APIs, and APIs associated with your task type. To learn more, see [Use APIs in Amazon Augmented AI \(p. 2394\)](#).

**Note**

Amazon Region availability may differ when you use Augmented AI with other Amazon services, such as Amazon Textract. Create Augmented AI resources in the same Amazon Region that you use to interact with those Amazon services. For Amazon Region availability for all services, see the [Region Table](#).

## Prerequisites to Using Augmented AI

Amazon A2I uses resources in IAM, SageMaker, and Amazon S3 to create and run your human review workflows. You can create some of these resources in the Amazon A2I console when you create a human review workflow. To learn how, see [Tutorial: Get Started in the Amazon A2I Console \(p. 2314\)](#).

To use Amazon A2I, you need the following resources:

- One or more Amazon S3 buckets in the same Amazon Region as the workflow for your input and output data. To create a bucket, follow the instructions in [Create a Bucket](#) in the *Amazon Simple Storage Service Console User Guide*.
- An IAM role with required permissions to create a human review workflow and an IAM user or role with permission to access Augmented AI. For more information, see [Permissions and Security in Amazon Augmented AI \(p. 2386\)](#).
- A public, private, or vendor workforce for your human review workflows. If you plan to use a private workforce, you need to set one up ahead of time in the same Amazon Region as your Amazon A2I workflow. To learn more about these workforce types, see [Create and Manage Workforces \(p. 456\)](#).

**Important**

To learn about the compliance programs that cover Amazon Augmented AI at this time, see [Amazon Services in Scope by Compliance Program](#). If you use Amazon Augmented AI in conjunction with other Amazon services (such as Amazon Rekognition and Amazon Textract), note that Amazon Augmented AI may not be in scope for the same compliance programs as those other services. You are responsible for how you use Amazon Augmented AI, including understanding how the service processes or stores customer data and any impact on the compliance of your data environment. You should discuss your workload objectives and goals with your Amazon account team; they can help you evaluate whether the service is a good fit for your proposed use case and architecture.

## Tutorial: Get Started in the Amazon A2I Console

The following tutorial shows you how to get started using Amazon A2I in the Amazon A2I console.

The tutorial gives you the option to use Augmented AI with Amazon Textract for document review or Amazon Rekognition for image content review.

### Prerequisites

To get started using Amazon A2I, complete the following prerequisites.

- Create an Amazon S3 bucket in the same Amazon Region as the workflow for your input and output data. For example, if you are using Amazon A2I with Amazon Textract in us-east-1, create your bucket in us-east-1. To create a bucket, follow the instructions in [Create a Bucket](#) in the *Amazon Simple Storage Service Console User Guide*.
- Do one of the following:
  - If you want to complete the tutorial using Amazon Textract, download [this sample document](#) and place it in your Amazon S3 bucket.

- If you want to complete the tutorial using Amazon Rekognition, download [this image](#) and place it in your Amazon S3 bucket.

**Note**

The Amazon A2I console is embedded in the SageMaker console.

## Step 1: Create a Work Team

First, create a work team in the Amazon A2I console and add yourself as a worker so that you can preview the worker review task.

**Important**

This tutorial uses a private work team. The Amazon A2I private workforce is configured in the Ground Truth area of the SageMaker console and is shared between Amazon A2I and Ground Truth.

### To create a private workforce using worker emails

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the navigation pane, choose **Labeling workforces** under **Ground Truth**.
3. Choose **Private**, then choose **Create private team**.
4. Choose **Invite new workers by email**.
5. For this tutorial, enter your email and any others that you want to be able to preview the human task UI. You can paste or type a list of up to 50 email addresses, separated by commas, into the email addresses box.
6. Enter an organization name and contact email.
7. Optionally, choose an Amazon SNS topic to which to subscribe the team so workers are notified by email when new Ground Truth labeling jobs become available. Amazon SNS notifications are supported by Ground Truth and are not supported by Augmented AI. If you subscribe workers to Amazon SNS notifications, they only receive notifications about Ground Truth labeling jobs. They do not receive notifications about Augmented AI tasks.
8. Choose **Create private team**.

If you add yourself to a private work team, you receive an email from `no-reply@verificationemail.com` with login information. Use the link in this email to reset your password and log in to your worker portal. This is where your human review tasks appear when you create a human loop.

## Step 2: Create a Human Review Workflow

In this step, you create a human review workflow. Each human review workflow is created for a specific [task type](#). This tutorial allows you to choose between the built-in task types: Amazon Rekognition and Amazon Textract.

### To create a human review workflow:

1. Open the Augmented AI console at <https://console.amazonaws.cn/a2i> to access the **Human review workflows** page.
2. Select **Create human review workflow**.
3. In **Workflow settings**, enter a workflow **Name**, **S3 bucket**, and the **IAM role** that you created for this tutorial, with the Amazon managed policy `AmazonAugmentedAIIntegratedAPIAccess` attached.
4. For **Task type**, select **Textract – Key-value pair extraction** or **Rekognition – Image moderation**.
5. Select the task type that you chose from the following table for instructions for that task type.

#### Amazon Textract – Key-value pair extraction

1. Select **Trigger a human review for specific form keys based on the form key confidence score or when specific form keys are missing.**
2. For **Key name**, enter **Mail Address**.
3. Set the identification confidence threshold between 0 and 99.
4. Set the qualification confidence threshold between 0 and 99.
5. Select **Trigger a human review for all form keys identified by Amazon Textract with confidence scores in a specific range.**
6. Set the identification confidence threshold between 0 and 90.
7. Set the qualification confidence threshold between 0 and 90.

This initiates a human review if Amazon Textract returns a confidence score that is less than 99 for **Mail Address** and its key, or if it returns a confidence score less than 90 for any key value pair detected in the document.

The following image shows the Amazon Textract form extraction - Conditions for invoking human review section of the Amazon A2I console. In the image, the check boxes for the two types of triggers explained in the proceeding paragraph are checked, and **Mail Address** is used as a **Key name** for the first trigger. The identification confidence threshold is defined using confidence scores for key-value pairs detect within the form and is set between 0 and 99. The qualification confidence threshold is defined using confidence scores for text contained within keys and values in a form and is set between 0 and 99.

**Amazon Textract form extraction - Conditions for invoking human review**

When Amazon Textract extracts information from a document, it returns a confidence score. You can use these confidence scores to define business conditions that trigger human review.

**Identification confidence**  
The confidence score for key-value pairs detected within a form.

**Qualification confidence**  
The confidence score for text contained within key and value in a form.

You can define a range for identification confidence and qualification confidence thresholds. A human review will be triggered when the confidence score falls within the defined range.

[Learn more about using Amazon Augmented AI with Amazon Textract](#)

Trigger a human review for specific form keys based on the form key confidence score or when specific form keys are missing. The form key and value will be sent to the human review.

**Key name**  
  
Trigger human review when this form key is missing,  
or when its identification confidence threshold is between  and   
or when its qualification confidence threshold is between  and   
[Add key](#)

Trigger human review for all form keys identified by Amazon Textract with confidence scores in a specified range. The form key and value will be sent to the human review.

**Identification confidence threshold**  
Trigger human review for key-value pairs detected within a form whose confidence score is in the following range:  
  and   
Minimum value is 0. Maximum value is 100.  
**Qualification confidence threshold**  
Trigger human review when the total confidence within key-value pairs in a form has confidence scores in the following range:  
  and   
Minimum value is 0. Maximum value is 100.  
 Randomly send a sample of forms to humans for review.  
For each form sent, all key-value pairs identified by Amazon Textract for that form will be sent for human review.

### Amazon Rekognition – Image moderation

1. Select **Trigger human review for labels identified by Amazon Rekognition based on label confidence score**.

2. Set the **Threshold** between 0 and 98.

This initiates a human review if Amazon Rekognition returns a confidence score that is less than 98 for an image moderation job.

The following image shows how you can select the **Trigger human review for labels identified by Amazon Rekognition based on label confidence score** option and enter a **Threshold** between 0 and 98 in the Amazon A2I console.



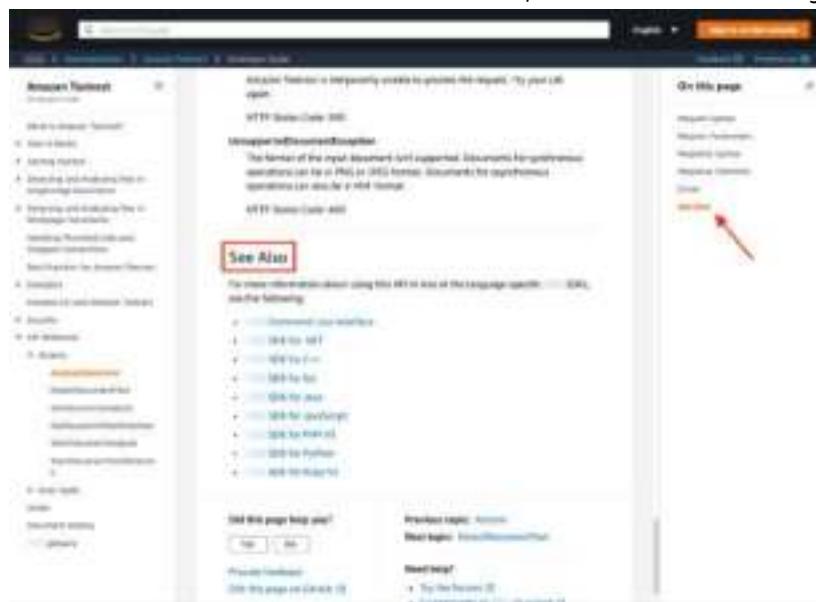
6. Under **Worker task template creation**, select **Create from a default template**.
7. Enter a **Template name**.
8. In **Task description** field, enter the following text:

Read the instructions carefully and complete the task.
9. Under **Workers**, select **Private**.
10. Select the private team that you created.
11. Choose **Create**.

Once your human review workflow is created, it appears in the table on the **Human review workflows** page. When the **Status** is **Active**, copy and save the Workflow ARN. You need it for the next step.

## Step 3: Start a Human Loop

You must use an API operation to start a human loop. There are a variety of language-specific SDKs that you can use to interact with these API operations. To see documentation for each of these SDKs, refer to the **See Also** section in the API documentation, as shown in the following image.



For this tutorial, you use one of the following APIs:

- If you chose the Amazon Textract task type, you use the [AnalyzeDocument](#) operation.
- If you chose the Amazon Rekognition task type, you use the [DetectModerationLabels](#) operation.

You can interact with these APIs using a SageMaker notebook instance (recommended for new users) or the Amazon Command Line Interface (Amazon CLI). Choose one of the following to learn more about these options:

- To learn more about and set up a notebook instance, see [Use Amazon SageMaker Notebook Instances \(p. 120\)](#).
- To learn more about and get started using the Amazon CLI, see [What Is the Amazon Command Line Interface?](#) in the *Amazon Command Line Interface User Guide*.

Select your task type in the following table to see example requests for Amazon Textract and Amazon Rekognition using the Amazon SDK for Python (Boto3).

#### Amazon Textract – Key-value pair extraction

The following example uses the Amazon SDK for Python (Boto3) to call `analyze_document` in us-west-2. Replace the italicized red text with your resources. Include the `DataAttributes` parameter if you are using the Amazon Mechanical Turk workforce. For more information, see the [analyze\\_document](#) documentation in the *Amazon SDK for Python (Boto) API Reference*.

```
response = client.analyze_document(
    Document={
        "S3Object": {
            "Bucket": "AWSDOC-EXAMPLE-BUCKET",
            "Name": "document-name.pdf"
        }
    },
    HumanLoopConfig={
        "FlowDefinitionArn": "arn:aws:sagemaker:us-west-2:111122223333:flow-
definition/flow-definition-name",
        "HumanLoopName": "human-loop-name",
        "DataAttributes": {
            "ContentClassifiers": [
                "FreeOfPersonallyIdentifiableInformation" | "FreeOfAdultContent"
            ]
        },
        FeatureTypes=[ "TABLES", "FORMS" ]
    }
)
```

#### Amazon Rekognition – Image moderation

The following example uses the Amazon SDK for Python (Boto3) to call `detect_moderation_labels` in us-west-2. Replace the italicized red text with your resources. Include the `DataAttributes` parameter if you are using the Amazon Mechanical Turk workforce. For more information, see the [detect\\_moderation\\_labels](#) documentation in the *Amazon SDK for Python (Boto) API Reference*.

```
response = client.detect_moderation_labels(
    Image={
        "S3Object": {
            "Bucket": "AWSDOC-EXAMPLE-BUCKET",
            "Name": "image-name.png"
        }
    },
    HumanLoopConfig={
        "FlowDefinitionArn": "arn:aws:sagemaker:us-west-2:111122223333:flow-
definition/flow-definition-name",
        "HumanLoopName": "human-loop-name",
        "DataAttributes": {
            "ContentClassifiers": [
                "FreeOfPersonallyIdentifiableInformation" | "FreeOfAdultContent"
            ]
        }
    }
)
```

```
}
```

## Step 4: View Human Loop Status in Console

When you start a human loop, you can view its status in the Amazon A2I console.

### To view your human loop status

1. Open the Augmented AI console at <https://console.amazonaws.cn/a2i> to access the **Human review workflows** page.
2. Select the human review workflow that you used to start your human loop.
3. In the **Human loops** section, you can see your human loop. View its status in the **Status** column.

## Step 5: Download Output Data

Your output data is stored in the Amazon S3 bucket you specified when you created a human review workflow.

### To view your Amazon A2I output data

1. Open the [Amazon S3 console](#).
2. Select the Amazon S3 bucket you specified when you created your human review workflow in step 2 of this example.
3. Starting with the folder that is named after your human review workflow, navigate to your output data by selecting the folder with the following naming convention:

```
s3://output-bucket-specified-in-human-review-workflow/human-review-workflow-
name/YYYY/MM/DD/hh/mm/ss/human-loop-name/output.json
```

4. Select `output.json` and choose **Download**.

## Tutorial: Get Started Using the Amazon A2I API

This tutorial explains the API operations you can use to get started using Amazon A2I.

To use a Jupyter Notebook to run these operations, select a Jupyter Notebook from [Use Cases and Examples Using Amazon A2I \(p. 2330\)](#) and use [Use SageMaker Notebook Instance with Amazon A2I Jupyter Notebook \(p. 2332\)](#) to learn how to use it in a SageMaker notebook instance.

To learn more about the API operations you can use with Amazon A2I, see [Use APIs in Amazon Augmented AI \(p. 2394\)](#).

## Create a Private Work Team

You can create a private work team and add yourself as a worker so that you can preview Amazon A2I.

If you are not familiar with Amazon Cognito, we recommend that you use the SageMaker console to create a private workforce and add yourself as a private worker. For instructions, see [Step 1: Create a Work Team \(p. 2315\)](#).

If you are familiar with Amazon Cognito, you can use the following instructions to create a private work team using the SageMaker API. After you create a work team, note the work team ARN (`workteamArn`).

To learn more about the private workforce and other available configurations, see [Use a Private Workforce \(p. 461\)](#).

### Create a private workforce

If you have not created a private workforce, you can do so using an [Amazon Cognito user pool](#). Make sure that you have added yourself to this user pool. You can create a private work team using the Amazon SDK for Python (Boto3) `create_workforce` function. For other language-specific SDKs, refer to the list in [CreateWorkforce](#).

```
response = client.create_workforce(  
    CognitoConfig={  
        "UserPool": "Pool_ID",  
        "ClientId": "app-client-id"  
    },  
    WorkforceName="workforce-name"  
)
```

### Create a private work team

After you have created a private workforce in the Amazon Region to configure and start your human loop, you can create a private work team using the Amazon SDK for Python (Boto3) `create_workteam` function. For other language-specific SDKs, refer to the list in [CreateWorkteam](#).

```
response = client.create_workteam(  
    WorkteamName="work-team-name",  
    WorkforceName= "workforce-name",  
    MemberDefinitions=[  
        {  
            "CognitoMemberDefinition": {  
                "UserPool": "<aws-region>_ID",  
                "UserGroup": "user-group",  
                "ClientId": "app-client-id"  
            },  
        }  
    ]  
)
```

Access your work team ARN as follows:

```
workteamArn = response[ "WorkteamArn" ]
```

### List private work teams in your account

If you have already created a private work team, you can list all work teams in a given Amazon Region in your account using the Amazon SDK for Python (Boto3) `list_workteams` function. For other language-specific SDKs, refer to the list in [ListWorkteams](#).

```
response = client.list_workteams()
```

If you have numerous work teams in your account, you may want to use `MaxResults`, `SortBy`, and `NameContains` to filter your results.

## Create a Human Review Workflow

You can create a human review workflow using the Amazon A2I [CreateFlowDefinition](#) operation. Before you create your human review workflow, you need to create a human task UI. You can do this with the [CreateHumanTaskUi](#) operation.

If you are using Amazon A2I with the Amazon Textract or Amazon Rekognition integrations, you can specify activation conditions using a JSON.

### Create a Human Task UI

If you are creating a human review workflow to be used with Amazon Textract or Amazon Rekognition integrations, you need to use and modify pre-made worker task template. For all custom integrations, you can use your own custom worker task template. Use the following table to learn how to create a human task UI using a worker task template for the two built-in integrations. Replace the template with your own to customize this request.

Amazon Textract – Key-value pair extraction

To learn more about this template, see [Custom Template Example for Amazon Textract \(p. 2369\)](#).

```
template = r"""
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
{% capture s3_uri %}http://s3.amazonaws.com/
{{ task.input.aiServiceRequest.document.s3Object.bucket }}/
{{ task.input.aiServiceRequest.document.s3Object.name }}{% endcapture %}
<crowd-form>
    <crowd-textract-analyze-document
        src="{{ s3_uri | grant_read_access }}"
        initial-value="{{ task.input.selectedAiServiceResponse.blocks }}"
        header="Review the key-value pairs listed on the right and correct them if they
don't match the following document."
        no-key-edit=""
        no-geometry-edit=""
        keys="{{ task.input.humanLoopContext.importantFormKeys }}"
        block-types='["KEY_VALUE_SET"]'
        <short-instructions header="Instructions">
            <p>Click on a key-value block to highlight the corresponding key-value pair in
the document.
            </p><p><br></p>
            <p>If it is a valid key-value pair, review the content for the value. If the
content is incorrect, correct it.
            </p><p><br></p>
            <p>The text of the value is incorrect, correct it.</p>
            <p>
            </p><p><br></p>
            <p>A wrong value is identified, correct it.</p>
            <p>
            </p><p><br></p>
            <p>If it is not a valid key-value relationship, choose No.</p>
            <p>
            </p><p><br></p>
            <p>If you can't find the key in the document, choose Key not found.</p>
            <p>
            </p><p><br></p>
            <p>If the content of a field is empty, choose Value is blank.</p>
            <p>
            </p><p><br></p>
            <p><strong>Examples</strong></p>
            <p>Key and value are often displayed next or below to each other.
            </p><p><br></p>
```

```

<p>Key and value displayed in one line.</p>
<p>
</p><p><br></p>
<p>Key and value displayed in two lines.</p>
<p>
</p><p><br></p>
<p>If the content of the value has multiple lines, enter all the text without
line break.
Include all value text even if it extends beyond the highlight box.</p>
<p></
p>
</short-instructions>
<full-instructions header="Instructions"></full-instructions>
</crowd-extract-analyze-document>
</crowd-form>
"""

```

### Amazon Rekognition – Image moderation

To learn more about this template, see [Custom Template Example for Amazon Rekognition \(p. 2371\)](#).

```

template = r"""
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
{# capture s3_uri %}http://s3.amazonaws.com/
{{ task.input.aiServiceRequest.image.s3Object.bucket }}/
{{ task.input.aiServiceRequest.image.s3Object.name }}{% endcapture %}

<crowd-form>
  <crowd-rekognition-detect-moderation-labels
    categories='[
      {% for label in task.input.selectedAiServiceResponse.moderationLabels %}
        {
          name: "{{ label.name }}",
          parentName: "{{ label.parentName }}",
        },
      {% endfor %}
    ]'
    src="{{ s3_uri | grant_read_access }}"
    header="Review the image and choose all applicable categories."
  >
    <short-instructions header="Instructions">
      <style>
        .instructions {
          white-space: pre-wrap;
        }
      </style>
      <p class="instructions">Review the image and choose all applicable categories.
      If no categories apply, choose None.
      </p>
    </short-instructions>
    <b>Nudity</b>
    Visuals depicting nude male or female person or persons

    <b>Partial Nudity</b>
    Visuals depicting covered up nudity, for example using hands or pose

    <b>Revealing Clothes</b>
    Visuals depicting revealing clothes and poses

    <b>Physical Violence</b>
    Visuals depicting violent physical assault, such as kicking or punching

    <b>Weapon Violence</b>

```

```
Visuals depicting violence using weapons like firearms or blades, such as shooting
<b>Weapons</b>
Visuals depicting weapons like firearms and blades
</short-instructions>

<full-instructions header="Instructions"></full-instructions>
</crowd-rekognition-detect-moderation-labels>
</crowd-form>""
```

## Custom Integration

The following is an example template that can be used in a custom integration. This template is used in this [notebook](#), demonstrating a custom integration with Amazon Comprehend.

```
template = r"""
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
    <crowd-classifier
        name="sentiment"
        categories='["Positive", "Negative", "Neutral", "Mixed"]'
        initial-value="{{ task.input.initialValue }}"
        header="What sentiment does this text convey?"
    >
        <classification-target>
            {{ task.input.taskObject }}
        </classification-target>

        <full-instructions header="Sentiment Analysis Instructions">
            <p><strong>Positive</strong> sentiment include: joy, excitement, delight</p>
            <p><strong>Negative</strong> sentiment include: anger, sarcasm, anxiety</p>
            <p><strong>Neutral</strong>: neither positive or negative, such as stating a fact</p>
            <p><strong>Mixed</strong>: when the sentiment is mixed</p>
        </full-instructions>

        <short-instructions>
            Choose the primary sentiment that is expressed by the text.
        </short-instructions>
    </crowd-classifier>
</crowd-form>
"""
```

Using the template specified above, you can create a template using the Amazon SDK for Python (Boto3) [create\\_human\\_task\\_ui](#) function. For other language-specific SDKs, refer to the list in [CreateHumanTaskUi](#).

```
response = client.create_human_task_ui(
    HumanTaskUiName="human-task-ui-name",
    UiTemplate={
        "Content": template
    }
)
```

This response element contains the human task UI ARN. Save this as follows:

```
humanTaskUiArn = response["HumanTaskUiArn"]
```

## Create JSON to specify activation conditions

For Amazon Textract and Amazon Rekognition built-in integrations, you can save activation conditions in a JSON object and use this in your `CreateFlowDefinition` request.

Next, select a tab to see example activation conditions you can use for these built-in integrations. For additional information about activation condition options, see [JSON Schema for Human Loop Activation Conditions in Amazon Augmented AI \(p. 2345\)](#).

### Amazon Textract – Key-value pair extraction

This example specifies conditions for specific keys (such as `Mail address`) in the document. If Amazon Textract's confidence falls outside of the thresholds set here, the document is sent to a human for review, with the specific keys that initiated the human loop prompted to the worker.

```
import json

humanLoopActivationConditions = json.dumps(
    {
        "Conditions": [
            {
                "Or": [
                    {
                        "ConditionType": "ImportantFormKeyConfidenceCheck",
                        "ConditionParameters": {
                            "ImportantFormKey": "Mail address",
                            "ImportantFormKeyAliases": ["Mail Address:", "Mail address:", "Mailing Add:", "Mailing Addresses"],
                            "KeyValueBlockConfidenceLessThan": 100,
                            "WordBlockConfidenceLessThan": 100
                        }
                    },
                    {
                        "ConditionType": "MissingImportantFormKey",
                        "ConditionParameters": {
                            "ImportantFormKey": "Mail address",
                            "ImportantFormKeyAliases": ["Mail Address:", "Mail address:", "Mailing Add:", "Mailing Addresses"]
                        }
                    },
                    {
                        "ConditionType": "ImportantFormKeyConfidenceCheck",
                        "ConditionParameters": {
                            "ImportantFormKey": "Phone Number",
                            "ImportantFormKeyAliases": ["Phone number:", "Phone No.:",
                                "Number:"],
                            "KeyValueBlockConfidenceLessThan": 100,
                            "WordBlockConfidenceLessThan": 100
                        }
                    },
                    {
                        "ConditionType": "ImportantFormKeyConfidenceCheck",
                        "ConditionParameters": {
                            "ImportantFormKey": "*",
                            "KeyValueBlockConfidenceLessThan": 100,
                            "WordBlockConfidenceLessThan": 100
                        }
                    },
                    {
                        "ConditionType": "ImportantFormKeyConfidenceCheck",
                        "ConditionParameters": {
                            "ImportantFormKey": "*",
                            "KeyValueBlockConfidenceLessThan": 100,
                            "WordBlockConfidenceLessThan": 100
                        }
                    }
                ]
            }
        ]
    }
)
```

```

        "KeyValueBlockConfidenceGreaterThanOrEqual": 0,
        "WordBlockConfidenceGreaterThanOrEqual": 0
    }
}
]
}
)

```

#### Amazon Rekognition – Image moderation

The human loop activation conditions used here are tailored towards Amazon Rekognition content moderation; they are based on the confidence thresholds for the `Suggestive` and `Female Swimwear Or Underwear` moderation labels.

```

import json

humanLoopActivationConditions = json.dumps(
{
    "Conditions": [
        {
            "Or": [
                {
                    "ConditionType": "ModerationLabelConfidenceCheck",
                    "ConditionParameters": {
                        "ModerationLabelName": "Suggestive",
                        "ConfidenceLessThan": 98
                    }
                },
                {
                    "ConditionType": "ModerationLabelConfidenceCheck",
                    "ConditionParameters": {
                        "ModerationLabelName": "Female Swimwear Or Underwear",
                        "ConfidenceGreaterThanOrEqual": 98
                    }
                }
            ]
        }
    ]
}
)

```

## Create a human review workflow

This section gives an example of the `CreateFlowDefinition` Amazon SDK for Python (Boto3) request using the resources created in the previous sections. For other language-specific SDKs, refer to the list in [CreateFlowDefinition](#). Use the tabs in the following table to see the requests to create a human review workflow for Amazon Textract and Amazon Rekognition built-in integrations.

#### Amazon Textract – Key-value pair extraction

If you use the built-in integration with Amazon Textract, you must specify "AWS/Textract/AnalyzeDocument/Forms/V1" for "AwsManagedHumanLoopRequestSource" in `HumanLoopRequestSource`.

```

response = client.create_flow_definition(
    FlowDefinitionName="human-review-workflow-name",
    HumanLoopRequestSource={

```

```

        "AwsManagedHumanLoopRequestSource": "AWS/Texttract/AnalyzeDocument/Forms/V1"
    },
    HumanLoopActivationConfig={
        "HumanLoopActivationConditionsConfig": {
            "HumanLoopActivationConditions": humanLoopActivationConditions
        }
    },
    HumanLoopConfig={
        "WorkteamArn": workteamArn,
        "HumanTaskUiArn": humanTaskUiArn,
        "TaskTitle": "Document entry review",
        "TaskDescription": "Review the document and instructions. Complete the
task",
        "TaskCount": 1,
        "TaskAvailabilityLifetimeInSeconds": 43200,
        "TaskTimeLimitInSeconds": 3600,
        "TaskKeywords": [
            "document review",
        ],
    },
    OutputConfig={
        "S3OutputPath": "s3://prefix/",
    },
    RoleArn="arn:aws:iam:<account-number>:role/<role-name>",
    Tags=[
        {
            "Key": "string",
            "Value": "string"
        },
    ],
)

```

#### Amazon Rekognition – Image moderation

If you use the built-in integration with Amazon Rekognition, you must specify "AWS/Rekognition/DetectModerationLabels/Image/V3" for "AwsManagedHumanLoopRequestSource" in HumanLoopRequestSource.

```

response = client.create_flow_definition(
    FlowDefinitionName="human-review-workflow-name",
    HumanLoopRequestSource={
        "AwsManagedHumanLoopRequestSource": "AWS/Rekognition/
DetectModerationLabels/Image/V3"
    },
    HumanLoopActivationConfig={
        "HumanLoopActivationConditionsConfig": {
            "HumanLoopActivationConditions": humanLoopActivationConditions
        }
    },
    HumanLoopConfig={
        "WorkteamArn": workteamArn,
        "HumanTaskUiArn": humanTaskUiArn,
        "TaskTitle": "Image content moderation",
        "TaskDescription": "Review the image and instructions. Complete the task",
        "TaskCount": 1,
        "TaskAvailabilityLifetimeInSeconds": 43200,
        "TaskTimeLimitInSeconds": 3600,
        "TaskKeywords": [
            "content moderation",
        ],
    },
    OutputConfig={
        "S3OutputPath": "s3://prefix/",
    }
)

```

```
        },
        RoleArn="arn:aws:iam::<account-number>:role/<role-name>",
        Tags=[  
            {  
                "Key": "string",  
                "Value": "string"  
            },  
        ]  
    )
```

## Custom Integration

If you use a custom integration, exclude the following parameters: `HumanLoopRequestSource`, `HumanLoopActivationConfig`.

```
response = client.create_flow_definition(  
    FlowDefinitionName="human-review-workflow-name",  
    HumanLoopConfig={  
        "WorkteamArn": workteamArn,  
        "HumanTaskUiArn": humanTaskUiArn,  
        "TaskTitle": "Image content moderation",  
        "TaskDescription": "Review the image and instructions. Complete the task",  
        "TaskCount": 1,  
        "TaskAvailabilityLifetimeInSeconds": 43200,  
        "TaskTimeLimitInSeconds": 3600,  
        "TaskKeywords": [  
            "content moderation",  
        ],  
    },  
    OutputConfig={  
        "S3OutputPath": "s3://prefix/",  
    },  
    RoleArn="arn:aws:iam::<account-number>:role/<role-name>",  
    Tags=[  
        {  
            "Key": "string",  
            "Value": "string"  
        },  
    ]  
)
```

After you create a human review workflow, you can retrieve the flow definition ARN from the response:

```
humanReviewWorkflowArn = response["FlowDefinitionArn"]
```

## Create a Human Loop

The API operation you use to start a human loop depends on the Amazon A2I integration you use.

- If you use the Amazon Textract built-in integration, you use the [AnalyzeDocument](#) operation.
- If you use the Amazon Rekognition built-in integration, you use the [DetectModerationLabels](#) operation.
- If you use a custom integration, you use the [StartHumanLoop](#) operation.

Select your task type in the following table to see example requests for Amazon Textract and Amazon Rekognition using the Amazon SDK for Python (Boto3).

### Amazon Textract – Key-value pair extraction

The following example uses the Amazon SDK for Python (Boto3) to call `analyze_document` in us-west-2. Replace the italicized red text with your resources. Include the `DataAttributes` parameter if you are using the Amazon Mechanical Turk workforce. For more information, see the `analyze_document` documentation in the *Amazon SDK for Python (Boto) API Reference*.

```
response = client.analyze_document(
    Document={"S3Object": {"Bucket": "AWSDOC-EXAMPLE-BUCKET", "Name": "document-name.pdf"},
              "HumanLoopConfig={"
                "FlowDefinitionArn": "arn:aws:sagemaker:us-west-2:111122223333:flow-definition/flow-definition-name",
                "HumanLoopName": "human-loop-name",
                "DataAttributes": {"ContentClassifiers": ["FreeOfPersonallyIdentifiableInformation"] | ["FreeOfAdultContent"]}
              }
              FeatureTypes=[ "FORMS" ]
            )
```

Human loops are only created if Amazon Textract's confidence for document analysis task meets the activation conditions you specified in your human review workflow. You can check the `response` element to determine if a human loop has been created. To see everything included in this response, see [HumanLoopActivationOutput](#).

```
if "HumanLoopArn" in analyzeDocumentResponse["HumanLoopActivationOutput"]:
    # A human loop has been started!
    print(f"A human loop has been started with ARN:
{analyzeDocumentResponse["HumanLoopActivationOutput"]["HumanLoopArn"]}"")
```

### Amazon Rekognition – Image moderation

The following example uses the Amazon SDK for Python (Boto3) to call `detect_moderation_labels` in us-west-2. Replace the italicized red text with your resources. Include the `DataAttributes` parameter if you are using the Amazon Mechanical Turk workforce. For more information, see the `detect_moderation_labels` documentation in the *Amazon SDK for Python (Boto) API Reference*.

```
response = client.detect_moderation_labels(
    Image={"S3Object": {"Bucket": "AWSDOC-EXAMPLE-BUCKET", "Name": "image-name.png"}},
              "HumanLoopConfig={"
                "FlowDefinitionArn": "arn:aws:sagemaker:us-west-2:111122223333:flow-definition/flow-definition-name",
                "HumanLoopName": "human-loop-name",
                "DataAttributes": {"ContentClassifiers": ["FreeOfPersonallyIdentifiableInformation"] | ["FreeOfAdultContent"]}
              }
            )
```

Human loops are only created if Amazon Rekognition's confidence for an image moderation task meets the activation conditions you specified in your human review workflow. You can check the `response` element to determine if a human loop has been created. To see everything included in this response, see [HumanLoopActivationOutput](#).

```
if "HumanLoopArn" in response["HumanLoopActivationOutput"]:
```

```

# A human loop has been started!
print(f"A human loop has been started with ARN:
{response[ "HumanLoopActivationOutput" ][ "HumanLoopArn" ]}")

```

### Custom Integration

The following example uses the Amazon SDK for Python (Boto3) to call `start_human_loop` in us-west-2. Replace the italicized red text with your resources. Include the `DataAttributes` parameter if you are using the Amazon Mechanical Turk workforce. For more information, see the [start\\_human\\_loop](#) documentation in the *Amazon SDK for Python (Boto) API Reference*.

```

response = client.start_human_loop(
    HumanLoopName= "human-loop-name",
    FlowDefinitionArn= "arn:aws:sagemaker:us-west-2:111122223333:flow-definition/
flow-definition-name",
    HumanLoopInput={"InputContent": inputContentJson},
    DataAttributes={"ContentClassifiers":
        [ "FreeOfPersonallyIdentifiableInformation" | "FreeOfAdultContent" ]
    }
)

```

This example stores input content in the variable `inputContentJson`. Assume that the input content contains two elements: a text blurb and sentiment (such as Positive, Negative, or Neutral), and it is formatted as follows:

```

inputContent = {
    "initialValue": sentiment,
    "taskObject": blurb
}

```

The keys `initialValue` and `taskObject` must correspond to the keys used in the liquid elements of the worker task template. Refer to the custom template in [Create a Human Task UI \(p. 2322\)](#) to see an example.

To create `inputContentJson`, do the following:

```

import json

inputContentJson = json.dumps(inputContent)

```

A human loop starts each time you call `start_human_loop`. To check the status of your human loop, use `describe_human_loop`:

```

human_loop_info = a2i.describe_human_loop(HumanLoopName="human-loop-name")
print(f"HumanLoop Status: {resp[ "HumanLoopStatus" ]}")
print(f"HumanLoop Output Destination: {resp[ "HumanLoopOutput" ]}")

```

## Use Cases and Examples Using Amazon A2I

You can use Amazon Augmented AI to incorporate a human review into your workflow for *built-in task types*, Amazon Textract and Amazon Rekognition, or your own custom tasks using a *custom task type*.

When you create a human review workflow using one of the built-in task types, you can specify conditions, such as confidence thresholds, that initiate a human review. The service (Amazon Rekognition

or Amazon Textract) creates a human loop on your behalf when these conditions are met and supplies your input data directly to Amazon A2I to send to human reviewers. To learn more about the built-in task types, use the following:

- [Use Amazon Augmented AI with Amazon Textract \(p. 2332\)](#)
- [Use Amazon Augmented AI with Amazon Rekognition \(p. 2335\)](#)

When you use a custom task type, you create and start a human loop using the Amazon A2I Runtime API. Use the custom task type to incorporate a human review workflow with other Amazon services or your own custom ML application.

- For more details, see [Use Amazon Augmented AI with Custom Task Types \(p. 2337\)](#)

The following table outlines a variety of Amazon A2I use cases that you can explore using SageMaker Jupyter Notebooks. To get started with a Jupyter Notebook, use the instructions in [Use SageMaker Notebook Instance with Amazon A2I Jupyter Notebook \(p. 2332\)](#). For more examples, see this [GitHub repository](#).

Use Case	Description	Task Type
<a href="#">Use Amazon A2I with Amazon Textract</a>	Have humans review single-page documents to review important form key-value pairs, or have Amazon Textract randomly sample and send documents from your dataset to humans for review.	Built-in
<a href="#">Use Amazon A2I with Amazon Rekognition</a>	Have humans review unsafe images for explicit adult or violent content if Amazon Rekognition returns a low confidence score, or have Amazon Rekognition randomly sample and send images from your dataset to humans for review.	Built-in
<a href="#">Use Amazon A2I with Amazon Comprehend</a>	Have humans review Amazon Comprehend inferences about text data such as sentiment analysis, text syntax, and entity detection.	Custom
<a href="#">Use Amazon A2I with Amazon Transcribe</a>	Have humans review Amazon Transcribe transcriptions of video or audio files. Use the results of transcription human review loops to create a custom vocabulary and improve future transcriptions of similar video or audio content.	Custom
<a href="#">Use Amazon A2I with Amazon Translate</a>	Have humans review low-confidence translations returned from Amazon Translate.	Custom

Use Case	Description	Task Type
Use Amazon A2I to review real-time ML inferences	Use Amazon A2I to review real-time, low-confidence inferences made by a model deployed to a SageMaker hosted endpoint and incrementally train your model using Amazon A2I output data.	Custom
Use Amazon A2I to review tabular data	Use Amazon A2I to integrate a human review loop into an ML application that uses tabular data.	Custom

#### Topics

- [Use SageMaker Notebook Instance with Amazon A2I Jupyter Notebook \(p. 2332\)](#)
- [Use Amazon Augmented AI with Amazon Textract \(p. 2332\)](#)
- [Use Amazon Augmented AI with Amazon Rekognition \(p. 2335\)](#)
- [Use Amazon Augmented AI with Custom Task Types \(p. 2337\)](#)

## Use SageMaker Notebook Instance with Amazon A2I Jupyter Notebook

For an end-to-end example that demonstrates how to integrate an Amazon A2I human review loop into a machine learning workflow, you can use a Jupyter Notebook from this [GitHub Repository](#) in a SageMaker notebook instance.

#### To use an Amazon A2I custom task type sample notebook in an Amazon SageMaker notebook instance:

1. If you do not have an active SageMaker notebook instance, create one by following the instructions in [Step 1: Create an Amazon SageMaker Notebook Instance \(p. 53\)](#).
2. When your notebook instance is active, choose **Open JupyterLab** to the right of the notebook instance's name. It may take a few moments for JupyterLab to load.
3. Choose the  icon to clone a GitHub repository into your workspace.
4. Enter the [amazon-a2i-sample-jupyter-notebooks](#) repository HTTPS URL.
5. Choose **CLONE**.
6. Open the notebook that you would like to run.
7. Follow the instructions in the notebook to configure your human review workflow and human loop and run the cells.
8. To avoid incurring unnecessary charges, when you are done with the demo, stop and delete your notebook instance in addition to any Amazon S3 buckets, IAM roles, and CloudWatch Events resources created during the walkthrough.

## Use Amazon Augmented AI with Amazon Textract

Amazon Textract enables you to add document text detection and analysis to your applications. Amazon Augmented AI (Amazon A2I) directly integrates with Amazon Textract's AnalyzeDocument API

operation. You can use `AnalyzeDocument` to analyze a document for relationships between detected items. When you add an Amazon A2I human review loop to an `AnalyzeDocument` request, Amazon A2I monitors the Amazon Textract results and sends a document to one or more human workers for review when the conditions specified in your flow definition are met. For example, if you want a human to review a specific key like `Full_name:` and their associated input values, you can create an activation condition that starts a human review any time the `Full_name:` key is detected or when the inference confidence for that key falls within a range that you specify.

The following image depicts the Amazon A2I built-in workflow with Amazon Textract. On the left, the resources that are required to create an Amazon Textract human review workflow are depicted: an Amazon S3 bucket, activation conditions, a worker task template, and a work team. These resources are used to create a human review workflow, or flow definition. An arrow points right to the next step in the workflow: using Amazon Textract to configure a human loop with the human review workflow. A second arrow points right from this step to the step in which activation conditions specified in the human review workflow are met. This initiates the creation of a human loop. On the right of the image, the human loop is depicted in three steps: 1) the worker UI and tools are generated and the task is made available to workers, 2) workers review input data, and finally, 3) results are saved in Amazon S3.



You can specify when Amazon Textract sends a task to a human worker for review when creating a human review workflow or flow definition by specifying *activation conditions*.

You can set the following activation conditions when using the Amazon Textract task type:

- Initiate a human review for specific form keys based on the form key confidence score.
- Initiate a human review when specific form keys are missing.
- Initiate human review for all form keys identified by Amazon Textract with confidence scores in a specified range.
- Randomly send a sample of forms to humans for review.

When your activation condition depends on form key confidence scores, you can use two types of prediction confidence to initiate human loops:

- **Identification confidence** – The confidence score for key-value pairs detected within a form.
- **Qualification confidence** – The confidence score for text contained within key and value in a form.

In the image in the following section, **Full Name: Jane Doe** is the key-value pair, **Full Name** is the key, and **Jane Doe** is the value.

You can set these activation conditions using the Amazon SageMaker console when you create a human review workflow, or by creating a JSON for human loop activation conditions and specifying this as input in the `HumanLoopActivationConditions` parameter of `CreateFlowDefinition` API operation. To learn how specify activation conditions in JSON format, see [JSON Schema for Human Loop Activation Conditions in Amazon Augmented AI \(p. 2345\)](#) and [Use Human Loop Activation Conditions JSON Schema with Amazon Textract \(p. 2347\)](#).

**Note**

When using Augmented AI with Amazon Textract, create Augmented AI resources in the same Amazon Region you use to call `AnalyzeDocument`.

## Get Started: Integrate a Human Review into an Amazon Textract Analyze Document Job

To integrate a human review into an Amazon Textract text detection and analysis job, you need to create a flow definition, and then use the Amazon Textract API to integrate that flow definition into your workflow. To learn how to create a flow definition using the SageMaker console or Augmented AI API, see the following topics:

- [Create a Human Review Workflow \(Console\) \(p. 2340\)](#)
- [Create a Human Review Workflow \(API\) \(p. 2341\)](#)

After you've created your flow definition, see [Using Augmented AI with Amazon Textract](#) to learn how to integrate your flow definition into your Amazon Textract task.

## End-to-End Example Using Amazon Textract and Amazon A2I

For an end-to-end example that demonstrates how to use Amazon Textract with Amazon A2I using the console, see [Tutorial: Get Started in the Amazon A2I Console \(p. 2314\)](#).

To learn how to use the Amazon A2I API to create and start a human review, you can use [Amazon Augmented AI \(Amazon A2I\) integration with Amazon Textract's Analyze Document \[Example\]](#) in a SageMaker Notebook instance. To get started, see [Use SageMaker Notebook Instance with Amazon A2I Jupyter Notebook \(p. 2332\)](#).

## A2I Textract Worker Console Preview

When they're assigned a review task in an Amazon Textract workflow, workers might see a user interface similar to the following:



You can customize this interface in the SageMaker console when you create your human review definition, or by creating and using a custom template. To learn more, see [Create and Manage Worker Task Templates \(p. 2366\)](#).

## Use Amazon Augmented AI with Amazon Rekognition

Amazon Rekognition makes it easy to add image analysis to your applications. The Amazon Rekognition DetectModerationLabels API operation is directly integrated with Amazon A2I so that you can easily create a human loop to review unsafe images, such as explicit adult or violent content. You can use DetectModerationLabels to configure a human loop using a flow definition ARN. This enables Amazon A2I to analyze predictions made by Amazon Rekognition and send results to a human for review to ensure they meet the conditions set in your flow definition.

The following image depicts the Amazon A2I built-in workflow with Amazon Rekognition. On the left, the resources that are required to create an Amazon Rekognition human review workflow are depicted: an Amazon S3 bucket, activation conditions, a worker task template, and a work team. These resources are used to create a human review workflow, or flow definition. An arrow points right to the next step in the workflow: using Amazon Rekognition to configure a human loop with the human review workflow. A second arrow points right from this step to the step in which activation conditions specified in the human review workflow are met. This initiates the creation of a human loop. On the right of the image, the human loop is depicted in three steps: 1) the worker UI and tools are generated and the task is made available to workers, 2) workers review input data, and finally, 3) results are saved in Amazon S3.



You can set the following activation conditions when using the Amazon Rekognition task type:

- Initiate human review for labels identified by Amazon Rekognition based on the label confidence score.
- Randomly send a sample of images to humans for review.

You can set these activation conditions using the Amazon SageMaker console when you create a human review workflow, or by creating a JSON for human loop activation conditions and specifying this as input in the `HumanLoopActivationConditions` parameter of the `CreateFlowDefinition` API operation. To learn how specify activation conditions in JSON format, see [JSON Schema for Human Loop Activation Conditions in Amazon Augmented AI \(p. 2345\)](#) and [Use Human Loop Activation Conditions JSON Schema with Amazon Rekognition \(p. 2353\)](#).

**Note**

When using Augmented AI with Amazon Rekognition, create Augmented AI resources in the same Amazon Region you use to call `DetectModerationLabels`.

## Get Started: Integrate a Human Review into an Amazon Rekognition Image Moderation Job

To integrate a human review into an Amazon Rekognition, see the following topics:

- [Create a Human Review Workflow \(Console\) \(p. 2340\)](#)
- [Create a Human Review Workflow \(API\) \(p. 2341\)](#)

After you've created your flow definition, see [Using Augmented AI with Amazon Rekognition](#) to learn how to integrate your flow definition into your Amazon Rekognition task.

## End-to-end Demo Using Amazon Rekognition and Amazon A2I

For an end-to-end example that demonstrates how to use Amazon Rekognition with Amazon A2I using the console, see [Tutorial: Get Started in the Amazon A2I Console \(p. 2314\)](#).

To learn how to use the Amazon A2I API to create and start a human review, you can use [Amazon Augmented AI \(Amazon A2I\) integration with Amazon Rekognition \[Example\]](#) in a SageMaker notebook instance. To get started, see [Use SageMaker Notebook Instance with Amazon A2I Jupyter Notebook \(p. 2332\)](#).

## A2I Rekognition Worker Console Preview

When they're assigned a review task in an Amazon Rekognition workflow, workers might see a user interface similar to the following:



You can customize this interface in the SageMaker console when you create your human review definition, or by creating and using a custom template. To learn more, see [Create and Manage Worker Task Templates \(p. 2366\)](#).

## Use Amazon Augmented AI with Custom Task Types

You can use Amazon Augmented AI (Amazon A2I) to incorporate a human review (human loop) into *any* machine learning workflow using the *custom task type*. This option gives you the most flexibility to customize the conditions under which your data objects are sent to humans for review, as well as the look and feel of your worker user interface.

When you use a custom task type, you create a custom human review workflow and specify the conditions under which a data object is sent for human review directly in your application.

The following image depicts the Amazon A2I custom workflow. A custom ML model is used to generate predictions. The client application filters these predictions using user-defined criteria and determines if a human review is required. If so, these predictions are sent to Amazon A2I for human review. Amazon A2I collects the results of human review in Amazon S3, which can be accessed by the client application. If the filter determines that no human review is needed, predictions can be fed directly to the client application.



Use the procedures on this page to learn how to integrate Amazon A2I into any machine learning workflow using the custom task type.

### Create a human loop using a flow definition, integrate it into your application, and monitor the results

1. Complete the Amazon A2I [Prerequisites to Using Augmented AI \(p. 2314\)](#). Note the following:
  - The path to the Amazon Simple Storage Service (Amazon S3) bucket or buckets where you store your input and output data.
  - The Amazon Resource Name (ARN) of an Amazon Identity and Access Management (IAM) role with required permissions attached.
  - (Optional) The ARN of your private workforce, if you plan to use one.
2. Using HTML elements, create a custom worker template which Amazon A2I uses to generate your worker task UI. To learn how to create a custom template, see [Create Custom Worker Task Templates \(p. 2368\)](#).
3. Use the custom worker template from step 2 to generate a worker task template in the Amazon SageMaker console. To learn how, see [Create a Worker Task Template \(p. 2367\)](#).

In the next step, you create a flow definition:

4. If you want to create a flow definition using the SageMaker API, note the ARN of this worker task template for the next step.
5. If you are creating a flow definition using the console, your template automatically appears in **Worker task template** section when you choose **Create human review workflow**.
6. When creating your flow definition, provide the path to your S3 buckets, your IAM role ARN, and your worker template.
  - To learn how to create a flow definition using the SageMaker `CreateFlowDefinition` API, see [Create a Human Review Workflow \(API\) \(p. 2341\)](#).
  - To learn how to create a flow definition using the SageMaker console, see [Create a Human Review Workflow \(Console\) \(p. 2340\)](#).
7. Configure your human loop using the [Amazon A2I Runtime API](#). To learn how, see [Create and Start a Human Loop \(p. 2358\)](#).
8. To control when human reviews are initiated in your application, specify conditions under which `StartHumanLoop` is called in your application. Human loop activation conditions, such as

confidence thresholds that initiate the human loop, are not available when using Amazon A2I with custom task types. Every `StartHumanLoop` invocation results in a human review.

Once you have started a human loop, you can manage and monitor your loops using the Amazon Augmented AI Runtime API and Amazon EventBridge (also known as Amazon CloudWatch Events). To learn more, see [Monitor and Manage Your Human Loop \(p. 2376\)](#).

## End-to-end Tutorial Using Amazon A2I Custom Task Types

For an end-to-end examples that demonstrates how to integrate Amazon A2I into a variety of ML workflows, see the table in [Use Cases and Examples Using Amazon A2I \(p. 2330\)](#). To get started using one of these notebooks, see [Use SageMaker Notebook Instance with Amazon A2I Jupyter Notebook \(p. 2332\)](#).

# Create a Human Review Workflow

Use an Amazon Augmented AI (Amazon A2I) *human review workflow*, or *flow definition*, to specify the following:

- For the Amazon Textract and Amazon Rekognition built-in task types, the conditions under which your human loop is called
- The workforce to which your tasks are sent
- The set of instructions that your workforce receives, which is called a *worker task template*
- The configuration of your worker tasks, including the number of workers that receive a task and time limits to complete tasks
- Where your output data is stored

You can create a human review workflow in the SageMaker console or using the SageMaker `CreateFlowDefinition` operation. You can build a worker task template using the console for Amazon Textract and Amazon Rekognition task types while creating your flow definition.

### Important

Human loop activation conditions, which initiate the human loop—for example, confidence thresholds—are not available for Amazon A2I custom task types. When using the console to create a flow definition for a custom task type, you can't specify activation conditions. When using the Amazon A2I API to create a flow definition for a custom task type, you can't set the `HumanLoopActivationConditions` attribute of the `HumanLoopActivationConditionsConfig` parameter. To control when human reviews are initiated, specify conditions under which `StartHumanLoop` is called in your custom application. In this case, every `StartHumanLoop` invocation results in a human review. For more information, see [Use Amazon Augmented AI with Custom Task Types \(p. 2337\)](#).

### Prerequisites

To create a human review workflow definition, you must have completed the prerequisites described in [Prerequisites to Using Augmented AI \(p. 2314\)](#).

If you use the API to create a flow definition for any task type, or if you use a custom task type when creating a flow definition in the console, first create a worker task template. For more information, see [Create and Manage Worker Task Templates \(p. 2366\)](#).

If you want to preview your worker task template while creating a flow definition for a built-in task type in the console, ensure that you grant the role that you use to create the flow definition permission to

access the Amazon S3 bucket that contains your template artifacts using a policy like the one described in [Enable Worker Task Template Previews \(p. 2390\)](#).

#### Topics

- [Create a Human Review Workflow \(Console\) \(p. 2340\)](#)
- [Create a Human Review Workflow \(API\) \(p. 2341\)](#)
- [JSON Schema for Human Loop Activation Conditions in Amazon Augmented AI \(p. 2345\)](#)

## Create a Human Review Workflow (Console)

Use this procedure to create a Amazon Augmented AI (Amazon A2I) human review workflow using the SageMaker console. If you are new to Amazon A2I, we recommend that you create a private work team using people in your organization, and use this work team's ARN when creating your flow definition. To learn how to set up a private workforce and create a work team, see [Create a Private Workforce \(Amazon SageMaker Console\) \(p. 462\)](#). If you have already set up a private workforce, see [Create a Work Team Using the SageMaker Console \(p. 466\)](#) to learn how to add a work team to that workforce.

If you are using Amazon A2I with one of the built-in task types, you can create worker instructions using a default worker task template provided by Augmented AI while creating a human review workflow in the console. To see samples of the default templates provided by Augmented AI, see the built-in task types in [Use Cases and Examples Using Amazon A2I \(p. 2330\)](#).

#### To create flow definition (console)

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the navigation pane, under the **Augmented AI** section, choose **Human review workflows** and then choose **Create human review workflow**.
3. In **Overview**, do the following:
  - a. For **Name**, enter a unique workflow name. The name must be lowercase, unique within the Amazon Region in your account, and can have up to 63 characters. Valid characters include: a-z, 0-9, and - (hyphen).
  - b. For **S3 location for output**, enter the S3 bucket where you want to store the human review results. The bucket must be located in the same Amazon Region as the workflow.
  - c. For **IAM role**, choose the role that has the required permissions. If you choose a built-in task type and want to preview your worker template in the console, provide a role with the type of policy described in [Enable Worker Task Template Previews \(p. 2390\)](#) attached.
4. For **Task type**, choose the task type that you want the human worker to perform.
5. If you chose the Amazon Rekognition or Amazon Textract task type, specify the conditions that invoke human review.
  - For Amazon Rekognition image moderation tasks, choose an inference confidence score threshold interval that initiates human review.
  - For Amazon Textract tasks, you can initiate a human review when specific form keys are missing or when form key detection confidence is low. You can also initiate a human review if, after evaluating all of the form keys in the text, confidence is lower than your required threshold for any form key. Two variables specify your confidence thresholds: **Identification confidence** and **Qualification confidence**. To learn more about these variables, see [Use Amazon Augmented AI with Amazon Textract \(p. 2332\)](#).
  - For both task types, you can randomly send a percentage of data objects (images or forms) and their labels to humans for review.
6. Configure and specify your worker task template:

- a. If you are using the Amazon Rekognition or Amazon Textract task type:
    - In the **Create template** section:
      - To create instructions for your workers using the Amazon A2I default template for Amazon Rekognition and Amazon Textract task types, choose **Build from a default template**.
      - If you choose **Build from a default template**, create your instructions under **Worker task design**:
        - Provide a **Template name** that is unique in the Amazon Region you are in.
        - In the **Instructions** section, provide detailed instructions on how to complete your task. To help workers achieve greater accuracy, provide good and bad examples.
        - (Optional) In **Additional instructions**, provide your workers with additional information and instructions.
    - For information on creating effective instructions, see [Creating Good Worker Instructions \(p. 2375\)](#).
      - To select a custom template that you've created, choose it from the **Template** menu and provide a **Task description** to briefly describe the task for your workers. To learn how to create a custom template, see [Create a Worker Task Template \(p. 2367\)](#).
  - b. If you are using the custom task type:
    - In the **Worker task template** section, select your template from the list. All of the templates that you have created in the SageMaker console appear in this list. To learn how to create a template for custom task types, see [Create and Manage Worker Task Templates \(p. 2366\)](#).
7. (Optional) Preview your worker template:
- For Amazon Rekognition and Amazon Textract task types, you have the option to choose **See a sample worker task** to preview your worker task UI.
- If you are creating a flow definition for a custom task type, you can preview your worker task UI using the `RenderUiTemplate` operation. For more information, see [Preview a Worker Task Template \(p. 2375\)](#).
8. For **Workers**, choose a workforce type.
  9. Choose **Create**.

## Next Steps

After you've created a human review workflow, it appears in the console under **Human review workflows**. To see your flow definition's Amazon Resource Name (ARN) and configuration details, choose the workflow by selecting its name.

If you are using a built-in task type, you can use the flow definition ARN to start a human loop using that Amazon service's API (for example, the Amazon Textract API). For custom task types, you can use the ARN to start a human loop using the Amazon Augmented AI Runtime API. To learn more about both options, see [Create and Start a Human Loop \(p. 2358\)](#).

## Create a Human Review Workflow (API)

To create a flow definition using the SageMaker API, you use the `CreateFlowDefinition` operation. After you complete the [Prerequisites to Using Augmented AI \(p. 2314\)](#), use the following procedure to learn how to use this API operation.

For an overview of the `CreateFlowDefinition` operation, and details about each parameter, see [CreateFlowDefinition](#).

### To create a flow definition (API)

1. For `FlowDefinitionName`, enter a unique name. The name must be unique within the Amazon Region in your account, and can have up to 63 characters. Valid characters include: a-z, 0-9, and - (hyphen).
2. For `RoleArn`, enter the ARN of the role that you configured to grant access to your data sources.
3. For `HumanLoopConfig`, enter information about the workers and what they should see. For information about each parameter in `HumanLoopConfig`, see [HumanLoopConfig](#).
4. (Optional) If you are using a built-in task type, provide conditions that initiate a human loop in `HumanLoopActivationConfig`. To learn how to create the input required for the `HumanLoopActivationConfig` parameter, see [JSON Schema for Human Loop Activation Conditions in Amazon Augmented AI \(p. 2345\)](#). If you do not specify conditions here, when you provide a flow definition to the Amazon service associated with a built-in task type (for example, Amazon Textract or Amazon Rekognition), that service sends every task to a human worker for review.

If you are using a custom task type, `HumanLoopActivationConfig` is disabled. To learn how to control when tasks are sent to human workers using a custom task type, see [Use Amazon Augmented AI with Custom Task Types \(p. 2337\)](#).

5. (Optional) If you are using a built-in task type, specify the integration source (for example, Amazon Rekognition or Amazon Textract) in the `HumanLoopRequestSource` parameter.
6. For `OutputConfig`, indicate where in Amazon Simple Storage Service (Amazon S3) to store the output of the human loop.
7. (Optional) Use `Tags` to enter key-value pairs to help you categorize and organize a flow definition. Each tag consists of a key and a value, both of which you define.

#### Amazon Textract – Key-value pair extraction

The following is an example of a request to create an Amazon Textract human review workflow (flow definition) using the Amazon SDK for Python (Boto3). You must use '`AWS/Textract/AnalyzeDocument/Forms/V1`' to create a Amazon Textract human loop. Only include `PublicWorkforceTaskPrice` if you are using the Mechanical Turk workforce.

```
sagemaker_client = boto3.client('sagemaker', aws_region)

response = sagemaker_client.create_flow_definition(
    FlowDefinitionName='ExampleFlowDefinition',
    HumanLoopRequestSource={
        'AwsManagedHumanLoopRequestSource': 'AWS/Textract/AnalyzeDocument/Forms/V1'
    },
    HumanLoopActivationConfig={
        'HumanLoopActivationConditionsConfig': {
            'HumanLoopActivationConditions': '{...}'
        }
    },
    HumanLoopConfig={
        'WorkteamArn': 'arn:aws:sagemaker:aws_region:aws_account_number:workteam/private-crowd/workteam_name',
        'HumanTaskUiArn': 'arn:aws:sagemaker:aws_region:aws_account_number:human-task-ui/template_name',
        'TaskTitle': 'Example task title',
        'TaskDescription': 'Example task description.',
        'TaskCount': 123,
        'TaskAvailabilityLifetimeInSeconds': 123,
        'TaskTimeLimitInSeconds': 123,
        'TaskKeywords': [
            'Keyword1', 'Keyword2'
        ]
    }
)
```

```

        ],
        'PublicWorkforceTaskPrice': {
            'AmountInUsd': {
                'Dollars': 123,
                'Cents': 123,
                'TenthFractionsOfACent': 123
            }
        }
    },
    OutputConfig={
        'S3OutputPath': 's3://bucket/path/',
        'KmsKeyId': '1234abcd-12ab-34cd-56ef-1234567890ab'
    },
    RoleArn='arn:aws:iam::aws_account_number:role/role_name',
    Tags:[
        {
            'Key': 'KeyName',
            'Value': 'ValueName'
        },
    ]
)

```

#### Amazon Rekognition – Image moderation

The following is an example of a request to create an Amazon Rekognition human review workflow (flow definition) using the Amazon SDK for Python (Boto3). You must use 'AWS/Rekognition/DetectModerationLabels/Image/V3' to create an Amazon Rekognition flow definition. Only include PublicWorkforceTaskPrice if you are using the Mechanical Turk workforce.

```

sagemaker_client = boto3.client('sagemaker', aws_region)

response = sagemaker_client.create_flow_definition(
    FlowDefinitionName='ExampleFlowDefinition',
    HumanLoopRequestSource={
        'AwsManagedHumanLoopRequestSource': 'AWS/Rekognition/DetectModerationLabels/
Image/V3'
    },
    HumanLoopActivationConfig={
        'HumanLoopActivationConditionsConfig': {
            'HumanLoopActivationConditions': '{...}'
        }
    },
    HumanLoopConfig={
        'WorkteamArn': 'arn:aws:sagemaker:aws_region:aws_account_number:workteam/
private-crowd/workteam_name',
        'HumanTaskUiArn': 'arn:aws:sagemaker:aws_region:aws_account_number:human-task-
ui/template_name',
        'TaskTitle': 'Example task title',
        'TaskDescription': 'Example task description.',
        'TaskCount': 123,
        'TaskAvailabilityLifetimeInSeconds': 123,
        'TaskTimeLimitInSeconds': 123,
        'TaskKeywords': [
            'Keyword1', 'Keyword2'
        ],
        'PublicWorkforceTaskPrice': {
            'AmountInUsd': {
                'Dollars': 123,
                'Cents': 123,
                'TenthFractionsOfACent': 123
            }
        }
    },
    OutputConfig=

```

```

        'S3OutputPath': 's3://bucket/path/',
        'KmsKeyId': '1234abcd-12ab-34cd-56ef-1234567890ab'
    },
    RoleArn='arn:aws:iam::aws_account_number:role/role_name',
    Tags=[  

        {  

            'Key': 'KeyName',  

            'Value': 'ValueName'  

        },  

    ]  

)

```

## Custom Workflow

The following is an example of a request to create a human review workflow (flow definition) for a custom integration. To create this type of human review workflow, omit `HumanLoopRequestSource` from the flow definition request. You only need to include `PublicWorkforceTaskPrice` if you are using the Mechanical Turk workforce.

```

sagemaker_client = boto3.client('sagemaker', aws_region)

response = sagemaker_client.create_flow_definition(  

    FlowDefinitionName='ExampleFlowDefinition',  

    HumanLoopActivationConfig={  

        'HumanLoopActivationConditionsConfig': {  

            'HumanLoopActivationConditions': '{...}'  

        }  

    },  

    HumanLoopConfig={  

        'WorkteamArn': 'arn:aws:sagemaker:aws_region:aws_account_number:workteam/  

private-crowd/workteam_name',  

        'HumanTaskUiArn': 'arn:aws:sagemaker:aws_region:aws_account_number:human-task-  

ui/template_name',  

        'TaskTitle': 'Example task title',  

        'TaskDescription': 'Example task description.',  

        'TaskCount': 123,  

        'TaskAvailabilityLifetimeInSeconds': 123,  

        'TaskTimeLimitInSeconds': 123,  

        'TaskKeywords': [  

            'Keyword1', 'Keyword2'  

        ],  

        'PublicWorkforceTaskPrice': {  

            'AmountInUsd': {  

                'Dollars': 123,  

                'Cents': 123,  

                'TenthFractionsOfACent': 123  

            }  

        }  

    },  

    OutputConfig={  

        'S3OutputPath': 's3://bucket/path/',
        'KmsKeyId': '1234abcd-12ab-34cd-56ef-1234567890ab'
    },
    RoleArn='arn:aws:iam::account_number:role/role_name',
    Tags=[  

        {  

            'Key': 'KeyName',  

            'Value': 'ValueName'  

        },
    ]
)

```

## Next Steps

The return value of a successful call of the `CreateFlowDefinition` API operation is a flow definition Amazon Resource Name (ARN).

If you are using a built-in task type, you can use the flow definition ARN to start a human loop using that Amazon service's API (i.e. the Amazon Textract API). For custom task types, you can use the ARN to start a human loop using the Amazon Augmented AI Runtime API. To learn more about both of these options, see [Create and Start a Human Loop \(p. 2358\)](#).

## JSON Schema for Human Loop Activation Conditions in Amazon Augmented AI

The `HumanLoopActivationConditions` is an input parameter of the `CreateFlowDefinition` API. This parameter is a JSON-formatted string. The JSON models the conditions under which a human loop is created when those conditions are evaluated against the response from an integrating AI service API (such as `Rekognition.DetectModerationLabels` or `Textract.AnalyzeDocument`). This response is referred to as an *inference*. For example, Amazon Rekognition sends an inference of a moderation label with an associated confidence score. In this example, the inference is the model's best estimate of the appropriate label for an image. For Amazon Textract, inference is made on the association between blocks of text (*key-value pairs*), such as the association between `Name` : and `Sue` in a form as well as content within a block of text, or *word block*, such as '`Name`'.

The following is the schema for the JSON. At the top level, the `HumanLoopActivationConditions` has a JSON array, `Conditions`. Each member of this array is an independent condition that, if evaluated to true, results in Amazon A2I creating a human loop. Each such independent condition can be a simple condition or a complex condition. A simple condition has the following attributes:

- `ConditionType`: This attribute identifies the type of condition. Each Amazon AI service API that integrates with Amazon A2I defines its own set of allowed `ConditionTypes`.
  - `Rekognition DetectModerationLabels` – This API supports the `ModerationLabelConfidenceCheck` and `Sampling ConditionType` values.
  - `Textract AnalyzeDocument` – This API supports the `ImportantFormKeyConfidenceCheck`, `MissingImportantFormKey`, and `Sampling ConditionType` values.
- `ConditionParameters` – This is a JSON object that parameterizes the condition. The set of allowed attributes of this object is dependent on the value of the `ConditionType`. Each `ConditionType` defines its own set of `ConditionParameters`.

A member of the `Conditions` array can model a complex condition. This is accomplished by logically connecting simple conditions using the `And` and `Or` logical operators and nesting the underlying simple conditions. Up to two levels of nesting are supported.

```
{  
    "$schema": "http://json-schema.org/draft-07/schema#",  
    "definitions": {  
        "Condition": {  
            "type": "object",  
            "properties": {  
                "ConditionType": {  
                    "type": "string"  
                },  
                "ConditionParameters": {  
                    "type": "object"  
                }  
            }  
        }  
    }  
}
```

```
        "required": [
            "ConditionType"
        ]
    },
    "OrConditionArray": {
        "type": "object",
        "properties": {
            "Or": {
                "type": "array",
                "minItems": 2,
                "items": {
                    "$ref": "#/definitions/ComplexCondition"
                }
            }
        }
    },
    "AndConditionArray": {
        "type": "object",
        "properties": {
            "And": {
                "type": "array",
                "minItems": 2,
                "items": {
                    "$ref": "#/definitions/ComplexCondition"
                }
            }
        }
    },
    "ComplexCondition": {
        "anyOf": [
            {
                "$ref": "#/definitions/Condition"
            },
            {
                "$ref": "#/definitions/OrConditionArray"
            },
            {
                "$ref": "#/definitions/AndConditionArray"
            }
        ]
    }
},
"type": "object",
"properties": {
    "Conditions": {
        "type": "array",
        "items": {
            "$ref": "#/definitions/ComplexCondition"
        }
    }
}
}
```

### Note

Human loop activation conditions aren't available for human review workflows that are integrated with custom task types. The `HumanLoopActivationConditions` parameter is disabled for custom task types.

### Topics

- [Use Human Loop Activation Conditions JSON Schema with Amazon Textract \(p. 2347\)](#)
- [Use Human Loop Activation Conditions JSON Schema with Amazon Rekognition \(p. 2353\)](#)

## Use Human Loop Activation Conditions JSON Schema with Amazon Textract

When used with Amazon A2I, the `AnalyzeDocument` operation supports the following inputs in the `ConditionType` parameter:

- `ImportantFormKeyConfidenceCheck` – Use this condition to create a human loop when inference confidence is within a specified range for document form keys and word blocks. A *form key* is any word in a document that is associated with an input. The input is called a *value*. Together, form keys and values are referred to as *key-value pairs*. A *word block* refers to the words that Amazon Textract recognizes inside of a detected block of text. To learn more about Amazon Textract document blocks, see [Documents and Block Objects](#) in the *Amazon Textract Developer Guide*.
- `MissingImportantFormKey` – Use this condition to create a human loop when Amazon Textract did not identify the key or its associated aliases within the document.
- `Sampling` – Use this condition to specify a percentage of forms to send to humans for review, regardless of inference confidence scores. Use this condition to do the following:
  - Audit your ML model by randomly sampling all forms analyzed by your model and sending a specified percentage to humans for review.
  - Using the `ImportantFormKeyConfidenceCheck` condition, randomly sample a percentage of the inferences that met the conditions specified in `ImportantFormKeyConfidenceCheck` to start a human loop and send only the specified percentage to humans for review.

### Note

If you send the same request to `AnalyzeDocument` multiple times, the result of `Sampling` does not change for the inference of that input. For example, if you make an `AnalyzeDocument` request once, and `Sampling` doesn't initiate a human loop, subsequent requests to `AnalyzeDocument` with the same configuration do not initiate a human loop.

## ImportantFormKeyConfidenceCheck Inputs and Results

The `ImportantFormKeyConfidenceCheck` `ConditionType` supports the following `ConditionParameters`:

- `ImportantFormKey` – A string representing a key in a key-value pair detected by Amazon Textract that needs to be reviewed by human workers. If the value of this parameter is the special catch-all value (\*), then all keys are considered to be matched to the condition. You can use this to model the case where any key-value pair satisfying certain confidence thresholds needs human review.
- `ImportantFormKeyAliases` – An array that represents alternate spellings or logical equivalents for the important form key.
- `KeyValueBlockConfidenceEquals`
- `KeyValueBlockConfidenceLessThan`
- `KeyValueBlockConfidenceLessThanEquals`
- `KeyValueBlockConfidenceGreaterThan`
- `KeyValueBlockConfidenceGreaterThanOrEqual`
- `WordBlockConfidenceEquals`
- `WordBlockConfidenceLessThan`
- `WordBlockConfidenceLessThanEquals`
- `WordBlockConfidenceGreaterThan`
- `WordBlockConfidenceGreaterThanOrEqual`

When you use the `ImportantFormKeyConfidenceCheck` ConditionType, Amazon A2I sends the key-value block and word block inferences of the key-value blocks and associated aliases that you specified in `ImportantFormKey` and `ImportantFormKeyAliases` for human review.

When creating a flow definition, if you use the default worker task template that is provided in the **Human review workflows** section of the Amazon SageMaker console, key-value and block inferences sent for human review by this activation condition are included in the worker UI. If you use a custom worker task template, you need to include the `{ { task.input.selectedAiServiceResponse.blocks } }` element to include initial-value input data (inferences) from Amazon Textract. For an example of a custom template that uses this input element, see [Custom Template Example for Amazon Textract \(p. 2369\)](#).

## MissingImportantFormKey Inputs and Results

The `MissingImportantFormKey` ConditionType supports the following ConditionParameters:

- `ImportantFormKey` – A string representing a key in a key-value pair detected by Amazon Textract that needs to be reviewed by human workers.
- `ImportantFormKeyAliases` – An array that represents alternate spellings or logical equivalents for the important form key.

When you use the `MissingImportantFormKey` ConditionType, if the key in `ImportantFormKey` or aliases in `ImportantFormKeyAliases` are not included in the Amazon Textract inference, that form is sent to human for review and no predicted key-value pairs are included. For example, if Amazon Textract only identified `Address` and `Phone` in a form, but was missing the `ImportantFormKey Name` (in the `MissingImportantFormKey` condition type) that form would be sent to humans for review without any of the form keys detected (`Address` and `Phone`).

If you use the default worker task template that is provided in the SageMaker console, a task is created asking workers to identify the key in `ImportantFormKey` and associated value. If you use a custom worker task template, you need to include the `<task.input.humanLoopContext>` custom HTML element to configure this task.

## Sampling Inputs and Results

The `Sampling` ConditionType supports the `RandomSamplingPercentage` ConditionParameters. The input for `RandomSamplingPercentage` must be real number between 0.01 and 100. This number represents the percentage of data that qualifies for a human review and is sent to humans for review. If you use the `Sampling` condition without any other conditions, this number represents the percentage of all resulting inferences made by the `AnalyzeDocument` operation from a single request that is sent to humans for review.

If you specify the `Sampling` condition without any other condition type, all key-value and block inferences are sent to workers for review.

When creating a flow definition, if you use the default worker task template that is provided in the **Human review workflows** section of the SageMaker console, all key-value and block inferences sent for human review by this activation condition are included in the worker UI. If you use a custom worker task template, you need to include the `{ { task.input.selectedAiServiceResponse.blocks } }` element to include initial-value input data (inferences) from Amazon Textract. For an example of a custom template that uses this input element, see [Custom Template Example for Amazon Textract \(p. 2369\)](#).

## Examples

While only one condition needs to evaluate to `true` to initiate a human loop, Amazon A2I evaluates all conditions for each object analyzed by Amazon Textract. The human reviewers are asked to review the important form keys for all the conditions that evaluated to `true`.

**Example 1: Detect important form keys with confidence scores in a specified range that initiate a human loop**

The following example shows a `HumanLoopActivationConditions` JSON that initiates a human loop if any one of the following three conditions is met:

- The Amazon Textract `AnalyzeDocument` API returns a key-value pair whose key is one of `EmployeeName`, `Name`, or `EmployeeName`, with the confidence of the key-value block being less than 60 and the confidences of each of the word blocks making up the key and value being less than 85.
- The Amazon Textract `AnalyzeDocument` API returns a key-value pair whose key is one of `Pay Date`, `PayDate`, `DateOfPay`, or `pay-date`, with the confidence of the key-value block being less than 65 and the confidences of each of the word blocks making up the key and value being less than 85.
- The Amazon Textract `AnalyzeDocument` API returns a key-value pair whose key is one of `Gross Pay`, `GrossPay`, or `GrossAmount`, with the confidence of the key-value block being less than 60 and the confidences of each of the word blocks making up the key and value being less than 85.

```
{
    "Conditions": [
        {
            "ConditionType": "ImportantFormKeyConfidenceCheck",
            "ConditionParameters": {
                "ImportantFormKey": "Employee Name",
                "ImportantFormKeyAliases": [
                    "Name",
                    "EmployeeName"
                ],
                "KeyValueBlockConfidenceLessThan": 60,
                "WordBlockConfidenceLessThan": 85
            }
        },
        {
            "ConditionType": "ImportantFormKeyConfidenceCheck",
            "ConditionParameters": {
                "ImportantFormKey": "Pay Date",
                "ImportantFormKeyAliases": [
                    "PayDate",
                    "DateOfPay",
                    "pay-date"
                ],
                "KeyValueBlockConfidenceLessThan": 65,
                "WordBlockConfidenceLessThan": 85
            }
        },
        {
            "ConditionType": "ImportantFormKeyConfidenceCheck",
            "ConditionParameters": {
                "ImportantFormKey": "Gross Pay",
                "ImportantFormKeyAliases": [
                    "GrossPay",
                    "GrossAmount"
                ],
                "KeyValueBlockConfidenceLessThan": 60,
                "WordBlockConfidenceLessThan": 85
            }
        }
    ]
}
```

**Example 2: Use `ImportantFormKeyConfidenceCheck`**

In the following example, if Amazon Textract detects a key-value pair whose confidence for the key-value block is less than 60 and is less than 90 for any underlying word blocks, it creates a human loop. The human reviewers are asked to review all the form key-value pairs that matched the confidence value comparisons.

```
{  
    "Conditions": [  
        {  
            "ConditionType": "ImportantFormKeyConfidenceCheck",  
            "ConditionParameters": {  
                "ImportantFormKey": "*"  
                "KeyValueBlockConfidenceLessThan": 60,  
                "WordBlockConfidenceLessThan": 90  
            }  
        }  
    ]  
}
```

### Example 3: Use Sampling

In the following example, 5% of inferences resulting from an Amazon Textract AnalyzeDocument request are sent to human workers for review. All detected key-value pairs returned by Amazon Textract are sent to workers for review.

```
{  
    "Conditions": [  
        {  
            "ConditionType": "Sampling",  
            "ConditionParameters": {  
                "RandomSamplingPercentage": 5  
            }  
        }  
    ]  
}
```

### Example 4: Use MissingImportantFormKey

In the following example, if `Mailing Address` or its alias, `Mailing Address:`, is missing from keys detected by Amazon Textract, a human review is initiated. When using the default worker task template, the worker UI asks workers to identify the key `Mailing Address` or `Mailing Address:` and its associated value.

```
{  
    "ConditionType": "MissingImportantFormKey",  
    "ConditionParameters": {  
        "ImportantFormKey": "Mailing Address",  
        "ImportantFormKeyAliases": ["Mailing Address:"]  
    }  
}
```

### Example 5: Use Sampling and ImportantFormKeyConfidenceCheck with the And operator

In this example, 5% of key-value pairs detected by Amazon Textract whose key is one of `Pay Date`, `PayDate`, `DateOfPay`, or `pay-date`, with the confidence of the key-value block less than 65 and the confidences of each of the word blocks making up the key and value less than 85, are sent to workers for review.

```
{
```

```

"Conditions": [
  {
    "And": [
      {
        "ConditionType": "Sampling",
        "ConditionParameters": {
          "RandomSamplingPercentage": 5
        }
      },
      {
        "ConditionType": "ImportantFormKeyConfidenceCheck",
        "ConditionParameters": {
          "ImportantFormKey": "Pay Date",
          "ImportantFormKeyAliases": [
            "PayDate",
            "DateOfPay",
            "pay-date"
          ],
          "KeyValueBlockConfidenceLessThan": 65,
          "WordBlockConfidenceLessThan": 85
        }
      }
    ]
  }
]
}

```

#### **Example 6: Use Sampling and ImportantFormKeyConfidenceCheck with the And operator**

Use this example to configure your human review workflow to always send low confidence inferences of a specified key-value pair for human review and sample high confidence inference of a key-value pair at a specified rate.

In the following example, a human review is initiated in one of the following ways:

- Key-value pairs detected whose key is one of Pay Date, PayDate, DateOfPay, or pay-date, with key-value and word block confidences less than 60, are sent for human review. Only the Pay Date form key (and its aliases) and associated values are sent to workers to review.
- 5% of key-value pairs detected whose key is one of Pay Date, PayDate, DateOfPay, or pay-date, with key-value and word block confidences greater than 90, are sent for human review. Only the Pay Date form key (and its aliases) and associated values are sent to workers to review.

```

{
  "Conditions": [
    {
      "Or": [
        {
          "ConditionType": "ImportantFormKeyConfidenceCheck",
          "ConditionParameters": {
            "ImportantFormKey": "Pay Date",
            "ImportantFormKeyAliases": [
              "PayDate",
              "DateOfPay",
              "pay-date"
            ],
            "KeyValueBlockConfidenceLessThan": 60,
            "WordBlockConfidenceLessThan": 60
          }
        },
        {
          "And": [
            {

```

```
        "ConditionType": "Sampling",
        "ConditionParameters": {
            "RandomSamplingPercentage": 5
        }
    },
{
    "ConditionType": "ImportantFormKeyConfidenceCheck",
    "ConditionParameters": {
        "ImportantFormKey": "Pay Date",
        "ImportantFormKeyAliases": [
            "PayDate",
            "DateOfPay",
            "pay-date"
        ],
        "KeyValueBlockConfidenceLessThan": 90
        "WordBlockConfidenceGreaterThan": 90
    }
}
]
}
]
```

**Example 7: Use Sampling and ImportantFormKeyConfidenceCheck with the Or operator**

In the following example, the Amazon Textract AnalyzeDocument operation returns a key-value pair whose key is one of Pay Date, PayDate, DateOfPay, or pay-date, with the confidence of the key-value block less than 65 and the confidences of each of the word blocks making up the key and value less than 85. Additionally, 5% of all other forms initiate a human loop. For each form randomly chosen, all key-value pairs detected for that form are sent to humans for review.

```
{
  "Conditions": [
    {
      "Or": [
        {
          "ConditionType": "Sampling",
          "ConditionParameters": {
            "RandomSamplingPercentage": 5
          }
        },
        {
          "ConditionType": "ImportantFormKeyConfidenceCheck",
          "ConditionParameters": {
            "ImportantFormKey": "Pay Date",
            "ImportantFormKeyAliases": [
              "PayDate",
              "DateOfPay",
              "pay-date"
            ],
            "KeyValueBlockConfidenceLessThan": 65,
            "WordBlockConfidenceLessThan": 85
          }
        }
      ]
    }
  ]
}
```

## Use Human Loop Activation Conditions JSON Schema with Amazon Rekognition

When used with Amazon A2I, the Amazon Rekognition `DetectModerationLabels` operation supports the following inputs in the `ConditionType` parameters:

- `ModerationLabelConfidenceCheck` – Use this condition type to create a human loop when inference confidence is low for one or more specified labels.
- `Sampling` – Use this condition to specify a percentage of all inferences to send to humans for review. Use this condition to do the following:
  - Audit your ML model by randomly sampling all of your model's inferences and sending a specified percentage to humans for review.
  - Using the `ModerationLabelConfidenceCheck` condition, randomly sample a percentage of the inferences that met the conditions specified in `ModerationLabelConfidenceCheck` to start a human loop and send only the specified percentage to humans for review.

### Note

If you send the same request to `DetectModerationLabels` multiple times, the result of `Sampling` does not change for the inference of that input. For example, if you make a `DetectModerationLabels` request once, and `Sampling` does not initiate a human loop, subsequent requests to `DetectModerationLabels` with the same configuration don't initiate a human loop.

When creating a flow definition, if you use the default worker task template that is provided in the **Human review workflows** section of the Amazon SageMaker console, inferences sent for human review by these activation conditions are included in the worker UI when a worker opens your task. If you use a custom worker task template, you need to include the `<task.input.selectedAiServiceResponse.blocks>` custom HTML element to access these inferences. For an example of a custom template that uses this HTML element, see [Custom Template Example for Amazon Rekognition \(p. 2371\)](#).

### ModerationLabelConfidenceCheck Inputs

For the `ModerationLabelConfidenceCheck` `ConditionType`, the following `ConditionParameters` are supported:

- `ModerationLabelName` – The exact (case-sensitive) name of a `ModerationLabel` detected by the Amazon Rekognition `DetectModerationLabels` operation. You can specify the special catch-all value (\*) to denote any moderation label.
- `ConfidenceEquals`
- `ConfidenceLessThan`
- `ConfidenceLessThanEquals`
- `ConfidenceGreater Than`
- `ConfidenceGreater ThanEquals`

When you use the `ModerationLabelConfidenceCheck` `ConditionType`, Amazon A2I sends label inferences for the labels that you specified in `ModerationLabelName` for human review.

### Sampling Inputs

The `Sampling` `ConditionType` supports the `RandomSamplingPercentage` `ConditionParameters`. The input for the `RandomSamplingPercentage` parameter should be real number between 0.01 and

100. This number represents the percentage of inferences that qualifies for a human review that are sent to humans for review. If you use the `Sampling` condition without any other conditions, this number represents the percentage of all inferences that result from a single `DetectModerationLabel` request that are sent to humans for review.

## Examples

### Example 1: Use `ModerationLabelConfidenceCheck` with the `And` operator

The following example of a `HumanLoopActivationConditions` condition initiates a human loop when one or more of the following conditions are met:

- Amazon Rekognition detects the `Graphic Male Nudity` moderation label with a confidence between 90 and 99.
- Amazon Rekognition detects the `Graphic Female Nudity` moderation label with a confidence between 80 and 99.

Note the use of the `Or` and `And` logical operators to model this logic.

Although only one of the two conditions under the `Or` operator needs to evaluate to `true` for a human loop to be created, Amazon Augmented AI evaluates all conditions. Human reviewers are asked to review the moderation labels for all the conditions that evaluated to `true`.

```
{
    "Conditions": [
        {
            "Or": [
                {
                    "And": [
                        {
                            "ConditionType": "ModerationLabelConfidenceCheck",
                            "ConditionParameters": {
                                "ModerationLabelName": "Graphic Male Nudity",
                                "ConfidenceLessThanEquals": 99
                            }
                        },
                        {
                            "ConditionType": "ModerationLabelConfidenceCheck",
                            "ConditionParameters": {
                                "ModerationLabelName": "Graphic Male Nudity",
                                "ConfidenceGreaterThanOrEqual": 90
                            }
                        }
                    ]
                },
                {
                    "And": [
                        {
                            "ConditionType": "ModerationLabelConfidenceCheck",
                            "ConditionParameters": {
                                "ModerationLabelName": "Graphic Female Nudity",
                                "ConfidenceLessThanEquals": 99
                            }
                        },
                        {
                            "ConditionType": "ModerationLabelConfidenceCheck",
                            "ConditionParameters": {
                                "ModerationLabelName": "Graphic Female Nudity",
                                "ConfidenceGreaterThanOrEqual": 80
                            }
                        }
                    ]
                }
            ]
        }
    ]
}
```

```
}
```

### Example 2: Use `ModerationLabelConfidenceCheck` with the catch-all value (\*)

In the following example, if any moderation label with a confidence greater than or equal to 75 is detected, a human loop is initiated. Human reviewers are asked to review all moderation labels with confidence scores greater than or equal to 75.

```
{
  "Conditions": [
    {
      "ConditionType": "ModerationLabelConfidenceCheck",
      "ConditionParameters": {
        "ModerationLabelName": "*",
        "ConfidenceGreaterThanOrEqual": 75
      }
    }
  ]
}
```

### Example 3: Use Sampling

In the following example, 5% of Amazon Rekognition inferences from a `DetectModerationLabels` request are sent to human workers. When using the default worker task template provided in the SageMaker console, all moderation labels returned by Amazon Rekognition are sent to workers for review.

```
{
  "Conditions": [
    {
      "ConditionType": "Sampling",
      "ConditionParameters": {
        "RandomSamplingPercentage": 5
      }
    }
  ]
}
```

### Example 4: Use Sampling and `ModerationLabelConfidenceCheck` with the `And` operator

In this example, 5% of Amazon Rekognition inferences of the `Graphic Male Nudity` moderation label with a confidence greater than 50 are sent workers for review. When using the default worker task template provided in the SageMaker console, only the inferences of the `Graphic Male Nudity` label are sent to workers for review.

```
{
  "Conditions": [
    {
      "And": [
        {
          "ConditionType": "Sampling",
          "ConditionParameters": {
            "RandomSamplingPercentage": 5
          }
        },
        {
          "ConditionType": "ModerationLabelConfidenceCheck",
          "ConditionParameters": {
            "ModerationLabelName": "Graphic Male Nudity",
            "ConfidenceGreaterThanOrEqual": 50
          }
        }
      ]
    }
  ]
}
```

```
        ]
    }
}
```

#### Example 5: Use Sampling and ModerationLabelConfidenceCheck with the And operator

Use this example to configure your human review workflow to always send low-confidence inferences of a specified label for human review and sample high-confidence inferences of a label at a specified rate.

In the following example, a human review is initiated in one of the following ways:

- Inferences for the `Graphic Male Nudity` moderation label with confidence scores less than 60 are always sent for human review. Only the `Graphic Male Nudity` label is sent to workers to review.
- 5% of all inferences for the `Graphic Male Nudity` moderation label with confidence scores greater than 90 are sent for human review. Only the `Graphic Male Nudity` label is sent to workers to review.

```
{
  "Conditions": [
    {
      "Or": [
        {
          "ConditionType": "ModerationLabelConfidenceCheck",
          "ConditionParameters": {
            "ModerationLabelName": "Graphic Male Nudity",
            "ConfidenceLessThan": 60
          }
        },
        {
          "And": [
            {
              "ConditionType": "Sampling",
              "ConditionParameters": {
                "RandomSamplingPercentage": 5
              }
            },
            {
              "ConditionType": "ModerationLabelConfidenceCheck",
              "ConditionParameters": {
                "ModerationLabelName": "Graphic Male Nudity",
                "ConfidenceGreaterThan": 90
              }
            }
          ]
        }
      ]
    }
}
```

#### Example 6: Use Sampling and ModerationLabelConfidenceCheck with the Or operator

In the following example, a human loop is created if the Amazon Rekognition inference response contains the 'Graphic Male Nudity' label with inference confidence greater than 50. Additionally, 5% of all other inferences initiate a human loop.

```
{
  "Conditions": [
```

```
{  
    "Or": [  
        {  
            "ConditionType": "Sampling",  
            "ConditionParameters": {  
                "RandomSamplingPercentage": 5  
            }  
        },  
        {  
            "ConditionType": "ModerationLabelConfidenceCheck",  
            "ConditionParameters": {  
                "ModerationLabelName": "Graphic Male Nudity",  
                "ConfidenceGreaterThan": 50  
            }  
        }  
    ]  
}
```

## Delete a Human Review Workflow

When you delete a human review workflow or you delete your Amazon account while a human loop is in process, your human review workflow status changes to `Deleting`. Amazon A2I automatically stops and deletes all associated human loops if workers have not started tasks created by those human loops. If human workers are already working on a task, that task continues to be available until it is completed or expires. As long as workers are still working on a task, your human review workflow's status is `Deleting`. If these tasks are completed, the results are stored in the Amazon S3 bucket specified in your flow definition.

Deleting a flow definition does not remove any worker answers from your S3 bucket. If the tasks are completed, but you deleted your Amazon account, the results are stored in the Augmented AI service bucket for 30 days and then permanently deleted.

After all human loops have been deleted, the human review workflow is permanently deleted. When a human review workflow has been deleted, you can reuse its name to create a new human review workflow.

You might want to delete a human review workflow for any of the following reasons:

- You have sent data to a set of human reviewers and you want to delete all non-started human loops because you do not want those workers to work on those tasks any longer.
- The worker task template used to generate your worker UI does not render correctly or is not functioning as expected.

After you delete a human review workflow, the following changes occur:

- The human review workflow no longer appears on the **Human review workflows** page in the Augmented AI area of the Amazon SageMaker console.
- When you use the human review workflow name as input to the API operations `DescribeFlowDefinition` or `DeleteFlowDefinition`, Augmented AI returns a `ResourceNotFoundException`.
- When you use `ListFlowDefinitions`, deleted human review workflows aren't included in the results.
- When you use the human review workflow ARN as input to the Augmented AI Runtime API operation `ListHumanLoops`, Augmented AI returns a `ResourceNotFoundException`.

# Delete a Flow Definition Using the Console or the SageMaker API

You can delete a human review workflow on the **Human review workflows** page in the Augmented AI area of the SageMaker console or by using the SageMaker API.

Flow definitions can only be deleted if their status is **Active**.

## Delete a human review workflow (console)

1. Navigate to the Augmented AI console at <https://console.amazonaws.cn/a2i/>.
2. In the navigation pane, under the **Augmented AI** section, choose **Human review workflows**.
3. Select the hyperlinked name of the human review workflow that you want to delete.
4. On the **Summary** page of your human review workflow, choose **Delete**.
5. In the dialog box asking you to confirm that you want to delete your human review workflow, choose **Delete**.

You're automatically redirected to the **Human review workflows** page. While your human review workflow is being deleted, the status **Deleting** appears in the status column for that workflow. After it's deleted, it doesn't appear in the list of workflows on this page.

## Delete a human review workflow (API)

You can delete a human review workflow (flow definition) using the SageMaker [DeleteFlowDefinition](#) API operation. This API operation is supported through the [Amazon CLI](#) and a [variety of language specific SDKs](#). The following table shows example requests using SDK for Python (Boto3) and the Amazon CLI to delete the human review workflow, *example-flow-definition*.

### Amazon SDK for Python (Boto3)

The following request example uses the SDK for Python (Boto3) to delete the human review workflow. For more information, see [delete\\_flow\\_definition](#) in the [Amazon SDK for Python \(Boto\) API Reference](#).

```
import boto3

sagemaker_client = boto3.client('sagemaker')
response = sagemaker_client.delete_flow_definition(FlowDefinitionName='example-flow-
definition')
```

### Amazon CLI

The following request example uses the Amazon CLI to delete the human review workflow. For more information, see [delete-flow-definition](#) in the [Amazon CLI Command Reference](#).

```
$ aws sagemaker delete-flow-definition --flow-definition-name 'example-flow-definition'
```

If the action is successful, Augmented AI sends back an HTTP 200 response with an empty HTTP body.

## Create and Start a Human Loop

A *human loop* starts your human review workflow and sends data review tasks to human workers. When you use one of the Amazon A2I built-in task types, the corresponding Amazon service creates and

starts a human loop on your behalf when the conditions specified in your flow definition are met. If no conditions are specified in your flow definition, a human loop is created for each object. When using Amazon A2I for a custom task, a human loop starts when your application calls `StartHumanLoop`.

Use the following instructions to configure a human loop with Amazon Rekognition or Amazon Textract built-in task types and custom task types.

### Prerequisites

To create and start a human loop, you must attach the `AmazonAugmentedAIFullAccess` policy to the Amazon Identity and Access Management (IAM) user or role that configures or starts the human loop. This is the identity that you use to configure the human loop using `HumanLoopConfig` for built-in task types. For custom task types, this is the identity that you use to call `StartHumanLoop`.

Additionally, when using a built-in task type, your IAM user or role must have permission to invoke API operations of the Amazon service associated with your task type. For example, if you are using Amazon Rekognition with Augmented AI, you must attach permissions required to call `DetectModerationLabels`. For examples of identity-based policies you can use to grant these permissions, see [Amazon Rekognition Identity-Based Policy Examples](#) and [Amazon Textract Identity-Based Policy Examples](#). You can also use the more general policy `AmazonAugmentedAIIIntegratedAPIAccess` to grant these permissions. For more information, see [Create an IAM User With Permissions to Invoke Amazon A2I, Amazon Textract, and Amazon Rekognition API Operations \(p. 2390\)](#).

To create and start a human loop, you need a flow definition ARN. To learn how to create a flow definition (or human review workflow), see [Create a Human Review Workflow \(p. 2339\)](#).

#### Important

Amazon A2I requires all S3 buckets that contain human loop input image data to have a CORS policy attached. To learn more about this change, see [CORS Permission Requirement \(p. 2387\)](#).

## Create and Start a Human Loop for a Built-in Task Type

To start a human loop using a built-in task type, use the corresponding service's API to provide your input data and to configure the human loop. For Amazon Textract, you use the `AnalyzeDocument` API operation. For Amazon Rekognition, you use the `DetectModerationLabels` API operation. You can use the Amazon CLI or a language-specific SDK to create requests using these API operations.

#### Important

When you create a human loop using a built-in task type, you can use `DataAttributes` to specify a set of `ContentClassifiers` related to the input provided to the `StartHumanLoop` operation. Use content classifiers to declare that your content is free of personally identifiable information or adult content.

To use Amazon Mechanical Turk, ensure your data is free of personally identifiable information, including protected health information under HIPAA. Include the `FreeOfPersonallyIdentifiableInformation` content classifier. If you do not use this content classifier, SageMaker does not send your task to Mechanical Turk. If your data is free of adult content, also include the '`FreeOfAdultContent`' classifier. If you do not use these content classifiers, SageMaker may restrict the Mechanical Turk workers that can view your task.

After you start your ML job using your built-in task type's Amazon service API, Amazon A2I monitors the inference results of that service. For example, when running a job with Amazon Rekognition, Amazon A2I checks the inference confidence score for each image and compares it to the confidence thresholds specified in your flow definition. If the conditions to start a human review task are satisfied, or if you didn't specify conditions in your flow definition, a human review task is sent to workers.

## Create an Amazon Textract Human Loop

Amazon A2I integrates with Amazon Textract so that you can configure and start a human loop using the Amazon Textract API. To send a document file to Amazon Textract for document analysis, you use the Amazon Textract [AnalyzeDocument API operation](#). To add a human loop to this document analysis job, you must configure the parameter `HumanLoopConfig`.

When you configure your human loop, the flow definition you specify in `FlowDefinitionArn` of `HumanLoopConfig` must be located in the same Amazon Region as the bucket identified in `Bucket` of the `Document` parameter.

The following table shows examples of how to use this operation with the Amazon CLI and Amazon SDK for Python (Boto3).

### Amazon SDK for Python (Boto3)

The following request example uses the SDK for Python (Boto3). For more information, see [analyze\\_document](#) in the *Amazon SDK for Python (Boto) API Reference*.

```
import boto3

textract = boto3.client('textract', aws_region)

response = textract.analyze_document(
    Document={'S3Object': {'Bucket': bucket_name, 'Name': document_name}},
    FeatureTypes=['TABLES', "FORMS"],
    HumanLoopConfig={
        'FlowDefinitionArn':
            'arn:aws:sagemaker:aws_region:aws_account_number:flow-definition/flow_def_name',
        'HumanLoopName': 'human_loop_name',
        'DataAttributes': {'ContentClassifiers':
            ['FreeOfPersonallyIdentifiableInformation', 'FreeOfAdultContent']}
    }
)
```

### Amazon CLI

The following request example uses the Amazon CLI. For more information, see [analyze-document](#) in the *Amazon CLI Command Reference*.

```
$ aws textract analyze-document \
--document '{"S3Object":{"Bucket":bucket_name,"Name":document_name}}' \
--human-loop-config \
HumanLoopName="human_loop_name",FlowDefinitionArn="arn:aws:sagemaker:aws-
region:aws_account_number:flow-
definition/
flow_def_name",DataAttributes='{ContentClassifiers=[ "FreeOfPersonallyIdentifiableInformation",
"FreeOfAdultContent"]}' \
--feature-types '[ "TABLES", "FORMS"]'
```

```
$ aws textract analyze-document \
--document '{"S3Object":{"Bucket":bucket_name,"Name":document_name}}' \
--human-loop-config \
{"HumanLoopName": "human_loop_name", "FlowDefinitionArn": "arn:aws:sagemaker:aws_region:aws_account_
definition/flow_def_name", "DataAttributes": {"ContentClassifiers":
["FreeOfPersonallyIdentifiableInformation", "FreeOfAdultContent"]}}' \
--feature-types '[ "TABLES", "FORMS"]'
```

After you run `AnalyzeDocument` with a human loop configured, Amazon A2I monitors the results from `AnalyzeDocument` and checks it against the flow definition's activation conditions. If the Amazon Textract inference confidence score for one or more key-value pairs meets the conditions for review, Amazon A2I starts a human review loop and includes the `HumanLoopActivationOutput` object in the `AnalyzeDocument` response.

## Create an Amazon Rekognition Human Loop

Amazon A2I integrates with Amazon Rekognition so that you can configure and start a human loop using the Amazon Rekognition API. To send images to Amazon Rekognition for content moderation, you use the Amazon Rekognition `DetectModerationLabels` API operation. To configure a human loop, set the `HumanLoopConfig` parameter when you configure `DetectModerationLabels`.

When you configure your human loop, the flow definition you specify in `FlowDefinitionArn` of `HumanLoopConfig` must be located in the same Amazon Region as the S3 bucket identified in `Bucket` of the `Image` parameter.

The following table shows examples of how to use this operation with the Amazon CLI and Amazon SDK for Python (Boto3).

### Amazon SDK for Python (Boto3)

The following request example uses the SDK for Python (Boto3). For more information, see [detect\\_moderation\\_labels](#) in the *Amazon SDK for Python (Boto) API Reference*.

```
import boto3

rekognition = boto3.client("rekognition", aws_region)

response = rekognition.detect_moderation_labels( \
    Image={'S3Object': {'Bucket': bucket_name, 'Name': image_name}}, \
    HumanLoopConfig={ \
        'HumanLoopName': 'human_loop_name', \
        'FlowDefinitionArn': \
            "arn:aws:sagemaker:aws_region:aws_account_number:flow-definition/flow_def_name" \
        'DataAttributes': {'ContentClassifiers': \
            ['FreeOfPersonallyIdentifiableInformation', 'FreeOfAdultContent']} \
    })
```

### Amazon CLI

The following request example uses the Amazon CLI. For more information, see [detect-moderation-labels](#) in the *Amazon CLI Command Reference*.

```
$ aws rekognition detect-moderation-labels \
--image "S3Object={Bucket='bucket_name',Name='image_name'}" \
--human-loop-config \
HumanLoopName="human_loop_name",FlowDefinitionArn="arn:aws:sagemaker:aws_region:aws_account_number:flow_definition/flow_def_name",DataAttributes='{ContentClassifiers=["FreeOfPersonallyIdentifiableInformation", "FreeOfAdultContent"]}'
```

```
$ aws rekognition detect-moderation-labels \
--image "S3Object={Bucket='bucket_name',Name='image_name'}" \
--human-loop-config \
'{"HumanLoopName": "human_loop_name", "FlowDefinitionArn": \
"arn:aws:sagemaker:aws_region:aws_account_number:flow-definition/flow_def_name", \
"DataAttributes": {"ContentClassifiers": ["FreeOfPersonallyIdentifiableInformation", \
"FreeOfAdultContent"]}}'
```

After you run `DetectModerationLabels` with a human loop configured, Amazon A2I monitors the results from `DetectModerationLabels` and checks it against the flow definition's activation conditions. If the Amazon Rekognition inference confidence score for an image meets the conditions for review, Amazon A2I starts a human review loop and includes the response element `HumanLoopActivationOutput` in the `DetectModerationLabels` response.

## Create and Start a Human Loop for a Custom Task Type

To configure a human loop for a custom human review task, use the `StartHumanLoop` operation within your application. This section provides an example of a human loop request using the Amazon SDK for Python (Boto3) and the Amazon Command Line Interface (Amazon CLI). For documentation on other language-specific SDKs that support `StartHumanLoop`, use the **See Also** section of [StartHumanLoop](#) in the Amazon Augmented AI Runtime API documentation. Refer to [Use Cases and Examples Using Amazon A2I \(p. 2330\)](#) to see examples that demonstrate how to use Amazon A2I with a custom task type.

### Prerequisites

To complete this procedure, you need:

- Input data formatted as a string representation of a JSON-formatted file
- The Amazon Resource Name (ARN) of your flow definition
- A flow definition ARN

### To configure the human loop

1. For `DataAttributes`, specify a set of `ContentClassifiers` related to the input provided to the `StartHumanLoop` operation. Use content classifiers to declare that your content is free of personally identifiable information or adult content.

To use Amazon Mechanical Turk, ensure your data is free of personally identifiable information, including protected health information under HIPAA, and include the `FreeOfPersonallyIdentifiableInformation` content classifier. If you do not use this content classifier, SageMaker does not send your task to Mechanical Turk. If your data is free of adult content, also include the '`FreeOfAdultContent`' classifier. If you do not use these content classifiers, SageMaker may restrict the Mechanical Turk workers that can view your task.

2. For `FlowDefinitionArn`, enter the Amazon Resource Name (ARN) of your flow definition.
3. For `HumanLoopInput`, enter your input data as a string representation of a JSON-formatted file. Structure your input data and custom worker task template so that your input data is properly displayed to human workers when you start your human loop. See [Preview a Worker Task Template \(p. 2375\)](#) to learn how to preview your custom worker task template.
4. For `HumanLoopName`, enter a name for the human loop. The name must be unique within the Region in your account and can have up to 63 characters. Valid characters are a-z, 0-9, and - (hyphen).

### To start a human loop

- To start a human loop, submit a request similar to the following examples using your preferred language-specific SDK.

#### Amazon SDK for Python (Boto3)

The following request example uses the SDK for Python (Boto3). For more information, see [Boto 3 Augmented AI Runtime](#) in the *Amazon SDK for Python (Boto) API Reference*.

```
import boto3

a2i_runtime_client = boto3.client('sagemaker-a2i-runtime')

response = a2i_runtime_client.start_human_loop(
    HumanLoopName='human_loop_name',
    FlowDefinitionArn='arn:aws:sagemaker:aws-region:xyz:flow-definition/flow_def_name',
    HumanLoopInput={
        'InputContent': '{"InputContent": "{\"prompt\": \"What is the answer?\"}"}'
    },
    DataAttributes={
        'ContentClassifiers': [
            'FreeOfPersonallyIdentifiableInformation' | 'FreeOfAdultContent',
        ]
    }
)
```

### Amazon CLI

The following request example uses the Amazon CLI. For more information, see [start-human-loop](#) in the [Amazon CLI Command Reference](#).

```
$ aws sagemaker-a2i-runtime start-human-loop
  --flow-definition-arn 'arn:aws:sagemaker:aws_region:xyz:flow-
definition/flow_def_name' \
  --human-loop-name 'human_loop_name' \
  --human-loop-input '{"InputContent": "{\"prompt\": \"What is the answer?\"}"}' \
  --data-attributes
  ContentClassifiers="FreeOfPersonallyIdentifiableInformation", "FreeOfAdultContent" \
```

When you successfully start a human loop by invoking `StartHumanLoop` directly, the response includes a `HumanLoopARN` and a `HumanLoopActivationResults` object which is set to `NULL`. You can use this the human loop name to monitor and manage your human loop.

## Next Steps:

After starting a human loop, you can manage and monitor it with the Amazon Augmented AI Runtime API and Amazon CloudWatch Events. To learn more, see [Monitor and Manage Your Human Loop \(p. 2376\)](#).

## Delete a Human Loop

When you delete a human loop, the status changes to `Deleting`. When the human loop is deleted, the associated human review task is no longer available to workers. You might want to delete a human loop in one of the following circumstances:

- The worker task template used to generate your worker user interface does not render correctly or is not functioning as expected.
- A single data object was accidentally sent to workers multiple times.
- You no longer need a data object reviewed by a human.

If the status of a human loop is `InProgress`, you must stop the human loop before deleting it. When you stop a human loop, the status changes to `Stopping` while it is being stopped. When the status changes to `Stopped`, you can delete the human loop.

If human workers are already working on a task when you stop the associated human loop, that task continues to be available until it is completed or expires. As long as workers are still working on a task, your human loop's status is **Stopping**. If these tasks are completed, the results are stored in the Amazon S3 bucket URI specified in your human review workflow. If the worker leaves the task without submitting work, it is stopped and the worker can't return to the task. If no worker has started working on the task, it is stopped immediately.

If you delete the Amazon account used to create the human loop, it is stopped and deleted automatically.

## Human Loop Data Retention and Deletion

When a human worker completes a human review task, the results are stored in the Amazon S3 output bucket you specified in the human review workflow used to create the human loop. Deleting or stopping a human loop does not remove any worker answers from your S3 bucket.

Additionally, Amazon A2I temporarily stores human loop input and output data internally for the following reasons:

- If you configure your human loops so that a single data object is sent to multiple workers for review, Amazon A2I does not write output data to your S3 bucket until all workers have completed the review task. Amazon A2I stores partial answers—answers from individual workers—internally so that it can write full results to your S3 bucket.
- If you report a low-quality human review result, Amazon A2I can investigate and respond to your issue.
- If you lose access to or delete the output S3 bucket specified in the human review workflow used to create a human loop, and the task has already been sent to one or more workers, Amazon A2I needs a place to temporarily store human review results.

Amazon A2I deletes this data internally 30 days after a human loop's status changes to one of the following: **Deleted**, **Stopped**, or **Completed**. In other words, data is deleted 30 days after the human loop has been completed, stopped, or deleted. Additionally, this data is deleted after 30 days if you close the Amazon account used to create associated human loops.

## Stop and Delete a Flow Definition Using the Console or the Amazon A2I API

You can stop and delete a human loop in the Augmented AI console or by using the SageMaker API. When the human loop has been deleted, the status changes to **Deleted**.

### Delete a human loop (console)

1. Navigate to the Augmented AI console at <https://console.amazonaws.cn/a2i/>.
2. In the navigation pane, under the **Augmented AI** section, choose **Human review workflows**.
3. Choose the hyperlinked name of the human review workflow you used to create the human loop you want to delete.
4. In the **Human loops** section at the bottom of the page, select the human loop you want to stop and delete.
5. If the human loop status is **Completed**, **Stopped**, or **Failed**, select **Delete**.

If the human loop **Status** is **InProgress**, select **Stop**. When the status changes to **Stopped**, select **Delete**.

## Delete a human loop (API)

1. Check the status of your human loop using the Augmented AI Runtime API operation [DescribeHumanLoop](#). See examples using this operation in the following table.

### Amazon SDK for Python (Boto3)

The following example uses the SDK for Python (Boto3) to describe the human loop named [\*example-human-loop\*](#). For more information, see [describe\\_human\\_loop](#) in the *Amazon SDK for Python (Boto) API Reference*.

```
import boto3

a2i_runtime_client = boto3.client('sagemaker-a2i-runtime')
response = a2i_runtime_client.describe_human_loop(HumanLoopName='example-human-loop')
human_loop_status = response['HumanLoopStatus']
print(f'example-human-loop status is: {human_loop_status}')
```

### Amazon CLI

The following example uses the Amazon CLI to describe the human loop named [\*example-human-loop\*](#). For more information, see [describe-human-loop](#) in the *Amazon CLI Command Reference*.

```
$ aws sagemaker-a2i-runtime describe-human-loop --human-loop-name 'example-human-loop'
```

2. If the flow definition status is Completed, Stopped, or Failed, delete the flow definition using the Augmented AI Runtime API operation [DeleteHumanLoop](#).

### Amazon SDK for Python (Boto3)

The following example uses the SDK for Python (Boto3) to delete the human loop named [\*example-human-loop\*](#). For more information, see [delete\\_human\\_loop](#) in the *Amazon SDK for Python (Boto) API Reference*.

```
import boto3

a2i_runtime_client = boto3.client('sagemaker-a2i-runtime')
response = a2i_runtime_client.delete_human_loop(HumanLoopName='example-human-loop')
```

### Amazon CLI

The following example uses the Amazon CLI to delete the human loop named [\*example-human-loop\*](#). For more information, see [delete-human-loop](#) in the *Amazon CLI Command Reference*.

```
$ aws sagemaker-a2i-runtime delete-human-loop --human-loop-name 'example-human-loop'
```

If the human loop status is InProgress, stop the human loop using [StopHumanLoop](#) and then use [DeleteHumanLoop](#) to delete it.

### Amazon SDK for Python (Boto3)

The following example uses the SDK for Python (Boto3) to describe the human loop named `example-human-loop`. For more information, see [stop\\_human\\_loop](#) in the *Amazon SDK for Python (Boto) API Reference*.

```
import boto3

a2i_runtime_client = boto3.client('sagemaker-a2i-runtime')
response = a2i_runtime_client.stop_human_loop(HumanLoopName='example-human-loop')
```

### Amazon CLI

The following example uses the Amazon CLI to describe the human loop named `example-human-loop`. For more information, see [stop-human-loop](#) in the *Amazon CLI Command Reference*.

```
$ aws sagemaker-a2i-runtime stop-human-loop --human-loop-name 'example-human-loop'
```

## Create and Manage Worker Task Templates

You can create a task user interface for your workers by creating a *worker task template*. A worker task template is an HTML file that is used to display your input data and instructions to help workers complete your task.

For Amazon Rekognition or Amazon Textract task types, you can customize a pre-made worker task template using a graphical user interface (GUI) and avoid interacting with HTML code. For this option, use the instructions in [Create a Human Review Workflow \(Console\) \(p. 2340\)](#) to create a human review workflow and customize your worker task template in the Amazon SageMaker console. Once you create a template using these instructions, it appears on the worker task templates page of the [Augmented AI console](#).

If you are creating a human review workflow for a custom task type, you must create a *custom worker task template* using HTML code. For more information, see [Create Custom Worker Task Templates \(p. 2368\)](#).

If you create your template using HTML, you must use this template to generate an Amazon A2I *human task UI Amazon Resource Name (ARN)* in the Amazon A2I console. This ARN has the following format: `arn:aws:sagemaker:<aws-region>:<aws-account-number>:human-task-ui/<template-name>`. This ARN is associated with a worker task template resource that you can use in one or more human review workflows (flow definitions).

Generate a human task UI ARN using a worker task template by following the instructions found in [Create a Worker Task Template \(p. 2367\)](#) or by using the `CreateHumanTaskUi` API operation.

### Topics

- [Create and Delete Worker Task Templates \(p. 2366\)](#)
- [Create Custom Worker Task Templates \(p. 2368\)](#)
- [Creating Good Worker Instructions \(p. 2375\)](#)

## Create and Delete Worker Task Templates

You can use a worker template to customize the interface and instructions that your workers see when working on your tasks. Use the instructions on this page to create a worker task template in the

Augmented AI area of the Amazon SageMaker console. A starter template is provided for Amazon Textract and Amazon Rekognition tasks. To learn how to customize your template using HTML crowd elements, see [Create Custom Worker Task Templates \(p. 2368\)](#).

When you create a worker template in the worker task templates page of the Augmented AI area of the SageMaker console, a worker task template ARN is generated. Use this ARN as the input to `HumanTaskUiArn` when you create a flow definition using the API operation [CreateFlowDefinition](#). You can choose this template when creating a human review workflow on the human review workflows page of the console.

If you are creating a worker task template resource for an Amazon Textract or Amazon Rekognition task type, you can preview the worker UI that is generated from your template on the worker task templates console page. You must attach the policy described in [Enable Worker Task Template Previews \(p. 2390\)](#) to the IAM role that you use to preview the template.

## Create a Worker Task Template

You can create a worker task template using the SageMaker console and using the SageMaker API operation [CreateHumanTaskUi](#).

### Create a worker task template (console)

1. Open the Amazon A2I console at <https://console.amazonaws.cn/a2i/>.
2. Under **Amazon Augmented AI** in the left navigation pane, choose **Worker task templates**.
3. Choose **Create template**.
4. In **Template name**, enter a unique name.
5. (Optional) Enter an **IAM role** that grants Amazon A2I the permissions necessary to call services on your behalf.
6. In **Template type**, choose a template type from the dropdown list. If you are creating a template for a **Textract-form extraction** or **Rekognition-image moderation** task, choose the appropriate option.
7. Enter your custom template elements as follows:
  - If you selected the Amazon Textract or Amazon Rekognition task template, the **Template editor** autopopulates with a default template that you can customize.
  - If you are using a custom template, enter your predefined template in the editor.
8. (Optional) To complete this step, you must provide an IAM role ARN with permission to read Amazon S3 objects that get rendered on your user interface in **Step 5**.

You can only preview your template if you are creating templates for Amazon Textract or Amazon Rekognition.

Choose **See preview** to preview the interface and instructions that workers see. This is an interactive preview. After you complete the sample task and choose **Submit**, you see the resulting output from the task that you just performed.

If you are creating a worker task template for a custom task type, you can preview your worker task UI using `RenderUiTemplate`. For more information, see [Preview a Worker Task Template \(p. 2375\)](#).

9. When you're satisfied with your template, choose **Create**.

After you've created your template, you can select that template when you create a human review workflow in the console. Your template also appears in the **Amazon Augmented AI** section of the SageMaker console under **Worker task templates**. Choose your template to view its ARN. Use this ARN when using the [CreateFlowDefinition](#) API operation .

### Create a worker task template using a worker task template (API)

To generate a worker task template using the SageMaker API operation [CreateHumanTaskUi](#), specify a name for your UI in `HumanTaskUiName` and input your HTML template in `Content` under `UiTemplate`. Find documentation on language-specific SDKs that support this API operation in the **See Also** section of the [CreateHumanTaskUi](#).

## Delete a Worker Task Template

Once you have created a worker task template, you can delete it using the SageMaker console or the SageMaker API operation [DeleteHumanTaskUi](#).

When you delete a worker task template, you are not able to use human review workflows (flow definitions) created using that template to start human loops. Any human loops that have already been created using the worker task template that you delete continue to be processed until completion and are not impacted.

### Delete a worker task template (console)

1. Open the Amazon A2I console at <https://console.amazonaws.cn/a2i/>.
2. Under Amazon Augmented AI in the left navigation pane, choose **Worker task templates**.
3. Select the template that you want to delete.
4. Select **Delete**.
5. A modal appears to confirm your choice. Select **Delete**.

### Delete a worker task template (API)

To delete a worker task template using the SageMaker API operation [DeleteHumanTaskUi](#), specify a name of your UI in `HumanTaskUiName`.

## Create Custom Worker Task Templates

*Crowd HTML Elements* are web components that provide a number of task widgets and design elements that you can tailor to the question you want to ask. You can use these crowd elements to create a custom worker template and integrate it with an Amazon Augmented AI (Amazon A2I) human review workflow to customize the worker console and instructions.

For a list of all HTML crowd elements available to Amazon A2I users, see [Crowd HTML Elements Reference \(p. 482\)](#). For examples of templates, see the [Amazon GitHub repository](#), which contains over 60 sample custom task templates.

## Develop Templates Locally

When in the console to test how your template processes incoming data, you can test the look and feel of your template's HTML and custom elements in your browser by adding the following code to the top of your HTML file.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
```

This loads the necessary code to render the custom HTML elements. Use this code if you want to develop your template's look and feel in your preferred editor instead of in the console.

This code won't parse your variables. You might want to replace them with sample content while developing locally.

## Use External Assets

Amazon Augmented AI custom templates enable you to embed external scripts and style sheets. For example, the following header embeds a text/css style sheet name `stylesheet` located at `https://www.example.com/my-enhancement-styles.css` into the custom template.

### Example

```
<script src="https://www.example.com/my-enhancement-script.js"></script>
<link rel="stylesheet" type="text/css" href="https://www.example.com/my-enhancement-
styles.css">
```

If you encounter errors, ensure that your originating server is sending the correct MIME type and encoding headers with the assets.

For example, the MIME and encoding type for remote scripts is `application/javascript;CHARSET=UTF-8`.

The MIME and encoding type for remote stylesheets is `text/css;CHARSET=UTF-8`.

## Track Your Variables

When building a custom template, you must add variables to it to represent the pieces of data that might change from task to task, or worker to worker. If you're starting with one of the sample templates, you need to make sure you're aware of the variables it already uses.

For example, for a custom template that integrates an Augmented AI human review loop with a Amazon Textract text review task, `{{ task.input.selectedAiServiceResponse.blocks }}` is used for initial-value input data. For Amazon Augmented AI (Amazon A2I) integration with Amazon Rekognition , `{{ task.input.selectedAiServiceResponse.moderationLabels }}` is used. For a custom task type, you need to determine the input parameter for your task type. Use `{{ task.input.customInputValuesForStartHumanLoop }}` where you specify `customInputValuesForStartHumanLoop`.

## Custom Template Example for Amazon Textract

All custom templates begin and end with the `<crowd-form> </crowd-form>` elements. Like standard HTML `<form>` elements, all of your form code should go between these elements.

For an Amazon Textract document analysis task, use the `<crowd-textract-analyze-document>` element. It uses the following attributes:

- `src` – Specifies the URL of the image file to be annotated.
- `initialValue` – Sets initial values for attributes found in the worker UI.
- `blockTypes` (required) – Determines the kind of analysis that the workers can do. Only `KEY_VALUE_SET` is currently supported.
- `keys` (required) – Specifies new keys and the associated text value that the worker can add.
- `no-key-edit` (required) – Prevents the workers from editing the keys of annotations passed through `initialValue`.
- `no-geometry-edit` – Prevents workers from editing the polygons of annotations passed through `initialValue`.

For children of the `<crowd-textract-analyze-document>` element, you must have two Regions. You can use arbitrary HTML and CSS elements in these Regions.

- <full-instructions> – Instructions that are available from the **View full instructions** link in the tool. You can leave this blank, but we recommend that you provide complete instructions to get better results.
- <short-instructions> – A brief description of the task that appears in the tool's sidebar. You can leave this blank, but we recommend that you provide complete instructions to get better results.

An Amazon Textract template would look similar to the following.

### Example

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
{% capture s3_uri %}http://s3.amazonaws.com/
{{ task.input.aiServiceRequest.document.s3Object.bucket }}/
{{ task.input.aiServiceRequest.document.s3Object.name }}{% endcapture %}

<crowd-form>
  <crowd-extract-analyze-document
    src="{{ s3_uri | grant_read_access }}"
    initial-value="{{ task.input.selectedAiServiceResponse.blocks }}"
    header="Review the key-value pairs listed on the right and correct them if they don't
match the following document."
    no-key-edit
    no-geometry-edit
    keys="{{ task.input.humanLoopContext.importantFormKeys }}"
    block-types="['KEY_VALUE_SET']"
  >
    <short-instructions header="Instructions">
      <style>
        .instructions {
          white-space: pre-wrap;
        }
        .instructionsImage {
          display: inline-block;
          max-width: 100%;
        }
      </style>
      <p class='instructions'>Choose a key-value block to highlight the corresponding key-
value pair in the document.

If it is a valid key-value pair, review the content for the value. If the content is
incorrect, correct it.

The text of the value is incorrect, correct it.


A wrong value is identified, correct it.


If it is not a valid key-value relationship, choose No.


If you can't find the key in the document, choose Key not found.


If the content of a field is empty, choose Value is blank.


<b>Examples</b>
Key and value are often displayed next to or below to each other.

Key and value displayed in one line.

```

```

Key and value displayed in two lines.


If the content of the value has multiple lines, enter all the text without a line break.
Include all value text even if it extends beyond the highlight box.
</p>
</short-instructions>

<full-instructions header="Instructions"></full-instructions>
</crowd-extract-analyze-document>
</crowd-form>

```

## Custom Template Example for Amazon Rekognition

All custom templates begin and end with the `<crowd-form>` `</crowd-form>` elements. Like standard HTML `<form>` elements, all of your form code should go between these elements. For an Amazon Rekognition custom task template, use the `<crowd-rekognition-detect-moderation-labels>` element. This element supports the following attributes:

- `categories` – An array of strings or an array of objects where each object has a `name` field.
  - If the `categories` come in as objects, the following applies:
    - The displayed categories are the value of the `name` field.
    - The returned answer contains the *full* objects of any selected categories.
  - If the `categories` come in as strings, the following applies:
    - The returned answer is an array of all the strings that were selected.
- `exclusion-category` – By setting this attribute, you create a button underneath the categories in the UI. When a user selects the button, all categories are deselected and disabled. If the worker selects the button again, you re-enable users to choose categories. If the worker submits the task by selecting **Submit** after you select the button, that task returns an empty array.

For children of the `<crowd-rekognition-detect-moderation-labels>` element, you must have two Regions.

- `<full-instructions>` – Instructions that are available from the **View full instructions** link in the tool. You can leave this blank, but we recommend that you provide complete instructions to get better results.
- `<short-instructions>` – Brief description of the task that appears in the tool's sidebar. You can leave this blank, but we recommend that you provide complete instructions to get better results.

A template using these elements would look similar to the following.

```

<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
{%
  capture s3_uri %}http://s3.amazonaws.com/
{{ task.input.aiServiceRequest.image.s3Object.bucket }}/
{{ task.input.aiServiceRequest.image.s3Object.name }}{% endcapture %}

<crowd-form>
  <crowd-rekognition-detect-moderation-labels
    categories='[
      {% for label in task.input.selectedAiServiceResponse.moderationLabels %}
        {
          name: "{{ label.name }}",
          parentName: "{{ label.parentName }}",
        },
        {% endfor %}
    ]'

```

```
src="{{ s3_uri | grant_read_access }}"
header="Review the image and choose all applicable categories."
>
<short-instructions header="Instructions">
  <style>
    .instructions {
      white-space: pre-wrap;
    }
  </style>
  <p class='instructions'>Review the image and choose all applicable categories.
  If no categories apply, choose None.

<b>Nudity</b>
Visuals depicting nude male or female person or persons

<b>Graphic Male Nudity</b>
Visuals depicting full frontal male nudity, often close ups

<b>Graphic Female Nudity</b>
Visuals depicting full frontal female nudity, often close ups

<b>Sexual Activity</b>
Visuals depicting various types of explicit sexual activities and pornography

<b>Illustrated Nudity or Sexual Activity</b>
Visuals depicting animated or drawn sexual activity, nudity, or pornography

<b>Adult Toys</b>
Visuals depicting adult toys, often in a marketing context

<b>Female Swimwear or Underwear</b>
Visuals depicting female person wearing only swimwear or underwear

<b>Male Swimwear Or Underwear</b>
Visuals depicting male person wearing only swimwear or underwear

<b>Partial Nudity</b>
Visuals depicting covered up nudity, for example using hands or pose

<b>Revealing Clothes</b>
Visuals depicting revealing clothes and poses, such as deep cut dresses

<b>Graphic Violence or Gore</b>
Visuals depicting prominent blood or bloody injuries

<b>Physical Violence</b>
Visuals depicting violent physical assault, such as kicking or punching

<b>Weapon Violence</b>
Visuals depicting violence using weapons like firearms or blades, such as shooting

<b>Weapons</b>
Visuals depicting weapons like firearms and blades

<b>Self Injury</b>
Visuals depicting self-inflicted cutting on the body, typically in distinctive patterns
using sharp objects

<b>Emaciated Bodies</b>
Visuals depicting extremely malnourished human bodies

<b>Corpses</b>
Visuals depicting human dead bodies

<b>Hanging</b>
Visuals depicting death by hanging</p>
```

```
</short-instructions>

<full-instructions header="Instructions"></full-instructions>
</crowd-rekognition-detect-moderation-labels>
</crowd-form>
```

## Add Automation with Liquid

The custom template system uses [Liquid](#) for automation. *Liquid* is an open-source inline markup language. For more information and documentation, see the [Liquid homepage](#).

In Liquid, the text between single curly braces and percent symbols is an instruction or *tag* that performs an operation like control flow or iteration. Text between double curly braces is a variable or *object* that outputs its value. The following list includes two types of liquid tags that you may find useful to automate template input data processing. If you select one of the following tag-types, you are redirected to the Liquid documentation.

- [Control flow](#): Includes programming logic operators like `if/else`, `unless`, and `case/when`.
- [Iteration](#): Enables you to run blocks of code repeatedly using statements like `for` loops.

For example, the following code example demonstrates how you can use the Liquid `for` tag to create a `for` loop. This example loops through the `moderationLabels` returned from Amazon Rekognition and displays the `moderationLabels` attributes `name` and `parentName` for workers to review:

```
{% for label in task.input.selectedAiServiceResponse.moderationLabels %}
{
    name: "{{ label.name }}",
    parentName: "{{ label.parentName }}",
},
{% endfor %}
```

## Use Variable Filters

In addition to the standard [Liquid filters](#) and actions, Amazon Augmented AI (Amazon A2I) offers additional filters. You apply filters by placing a pipe (|) character after the variable name, and then specifying a filter name. To chain filters, use the following format.

### Example

```
{ {<content> | <filter> | <filter> } }
```

### Autoescape and Explicit Escape

By default, inputs are HTML-escaped to prevent confusion between your variable text and HTML. You can explicitly add the `escape` filter to make it more obvious to someone reading the source of your template that escaping is being done.

#### `escape_once`

`escape_once` ensures that if you've already escaped your code, it doesn't get re-escaped again. For example, it ensures that `&` doesn't become `&amp;`.

#### `skip_autoescape`

`skip_autoescape` is useful when your content is meant to be used as HTML. For example, you might have a few paragraphs of text and some images in the full instructions for a bounding box.

### Note

Use `skip_autoescape` sparingly. As a best practice for templates, avoid passing in functional code or markup with `skip_autoescape` unless you are absolutely sure that you have strict control over what's being passed. If you're passing user input, you could be opening your workers up to a cross-site scripting attack.

### [to\\_json](#)

`to_json` encodes data that you provide to JavaScript Object Notation (JSON). If you provide an object, it serializes it.

### [grant\\_read\\_access](#)

`grant_read_access` takes an Amazon Simple Storage Service (Amazon S3) URI and encodes it into an HTTPS URL with a short-lived access token for that resource. This makes it possible to display photo, audio, or video objects stored in S3 buckets that are not otherwise publicly accessible to workers.

### **Example Example of the `to_json` and `grant_read_access` filters**

#### Input

```
auto-escape: {{ "Have you read 'James & the Giant Peach'?" }}
```

```
explicit escape: {{ "Have you read 'James & the Giant Peach'?" | escape }}
```

```
explicit escape_once: {{ "Have you read 'James & the Giant Peach'?" | escape_once }}
```

```
skip_autoescape: {{ "Have you read 'James & the Giant Peach'?" | skip_autoescape }}
```

```
to_json: {{ jsObject | to_json }}
```

```
grant_read_access: {{ "s3://examplebucket/myphoto.png" | grant_read_access }}
```

#### Example

#### Output

```
auto-escape: Have you read 'James & the Giant Peach'?
```

```
explicit escape: Have you read 'James & the Giant Peach'?
```

```
explicit escape_once: Have you read 'James & the Giant Peach'?
```

```
skip_autoescape: Have you read 'James & the Giant Peach'?
```

```
to_json: { "point_number": 8, "coords": [ 59, 76 ] }
```

```
grant_read_access: https://s3.amazonaws.com/examplebucket/myphoto.png?<access token and other params>
```

### **Example Example of an automated classification template.**

To automate this simple text classification sample, include the Liquid tag `{{ task.input.source }}`. This example uses the [crowd-classifier \(p. 496\)](#) element.

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>
<crowd-form>
  <crowd-classifier
    name="tweetFeeling"
    categories="['positive', 'negative', 'neutral', 'cannot determine']"
    header="Which term best describes this tweet?"
  >
    <classification-target>
      {{ task.input.source }}
    </classification-target>

    <full-instructions header="Analyzing a sentiment">
      Try to determine the feeling the author
    </full-instructions>
  </crowd-classifier>
</crowd-form>
```

```
of the tweet is trying to express.  
If none seems to match, choose "other."  
</full-instructions>  
  
<short-instructions>  
Pick the term that best describes the sentiment  
of the tweet.  
</short-instructions>  
  
</crowd-classifier>  
</crowd-form>
```

## Preview a Worker Task Template

To preview a custom worker task template, use the SageMaker `RenderUiTemplate` operation. You can use the `RenderUiTemplate` operation with the Amazon CLI or your preferred Amazon SDK. For documentation on the supported language specific SDKs for this API operation, see the [See Also](#) section of the [RenderUiTemplate](#).

### Prerequisites

To preview your worker task template, the Amazon Identity and Access Management (IAM) role Amazon Resource Name (ARN), or `RoleArn`, that you use must have permission to access to the S3 objects that are used by the template. To learn how to configure your role or user see [Enable Worker Task Template Previews \(p. 2390\)](#).

### To preview your worker task template using the `RenderUiTemplate` operation:

1. Provide a `RoleArn` of the role with required policies attached to preview your custom template.
2. In the `Input` parameter of `Task`, provide a JSON object that contains values for the variables defined in the template. These are the variables that are substituted for the `task.input.source` variable. For example, if you define a `task.input.text` variable in your template, you can supply the variable in the JSON object as `text: sample text`.
3. In the `Content` parameter of `UiTemplate`, insert your template.

Once you've configured `RenderUiTemplate`, use your preferred SDK or the Amazon CLI to submit a request to render your template. If your request was successful, the response includes `RenderedContent`, a Liquid template that renders the HTML for the worker UI.

### Important

To preview your template, you need an IAM role with permissions to read Amazon S3 objects that get rendered on your user interface. For a sample policy that you can attach to your IAM role to grant these permissions, see [Enable Worker Task Template Previews \(p. 2390\)](#).

## Creating Good Worker Instructions

Creating good instructions for your human review jobs improves your worker's accuracy in completing their task. You can modify the default instructions that are provided in the console when creating a human review workflow, or you can use the console to create a custom worker template and include your instructions in this template. The instructions are shown to the worker on the UI page where they complete their labeling task.

## Create Good Worker Instructions

There are three kinds of instructions in the Amazon Augmented AI console:

- **Task Description** – The description should provide a succinct explanation of the task.
- **Instructions** – These instructions are shown on the same webpage where workers complete a task. These instructions should provide an easy reference to show the worker the correct way to complete the task.
- **Additional Instructions** – These instructions are shown in a dialog box that appears when a worker chooses **View full instructions**. We recommend that you provide detailed instructions for completing the task, and include several examples showing edge cases and other difficult situations for labeling objects.

## Add Example Images to Your Instructions

Images provide useful examples for your workers. To add a publicly accessible image to your instructions, do the following:

1. Place the cursor where the image should go in the instructions editor.
2. Choose the image icon in the editor toolbar.
3. Enter the URL of your image.

If your instruction image is in an S3 bucket that isn't publicly accessible, do the following:

- For the image URL, enter: `{} 'https://s3.amazonaws.com/your-bucket-name/image-file-name' | grant_read_access {}`.

This renders the image URL with a short-lived, one-time access code that's appended so the worker's browser can display it. A broken image icon is displayed in the instructions editor, but previewing the tool displays the image in the rendered preview. See [grant\\_read\\_access \(p. 2374\)](#) for more information about the `grant_read_access` element.

## Monitor and Manage Your Human Loop

Once you've started a human review loop, you can check the results of tasks sent to the loop and manage it using the [Amazon Augmented AI Runtime API](#). Additionally, Amazon A2I integrates with Amazon EventBridge (also known as Amazon CloudWatch Events) to alert you when a human review loop status changes to `Completed`, `Failed`, or `Stopped`. This event delivery is guaranteed at least once, which means all events created when human loops finish are successfully delivered to EventBridge.

Use the procedures below to learn how to use the Amazon A2I Runtime API to monitor and manage your human loops. See [Use Amazon CloudWatch Events in Amazon Augmented AI \(p. 2392\)](#) to learn how Amazon A2I integrates with Amazon EventBridge.

### To check your output data:

1. Check the results of your human loop by calling the `DescribeHumanLoop` operation. The result of this API operation contains information about the reason for and outcome of the loop activation.
2. Check the output data from your human loop in Amazon Simple Storage Service (Amazon S3). In the path to the data, `YYYY/MM/DD/hh/mm/ss` represents the human loop creation date with year (`YYYY`), month (`MM`), and day (`DD`), and the creation time with hour (`hh`), minute (`mm`), and second (`ss`).

```
s3://customer-output-bucket-specified-in-flow-definition/flow-definition-  
name/YYYY/MM/DD/hh/mm/ss/human-loop-name/output.json
```

You can integrate this structure with Amazon Glue or Amazon Athena to partition and analyze your output data. For more information, see [Managing Partitions for ETL Output in Amazon Glue](#).

To learn more about Amazon A2I output data format, see [Amazon A2I Output Data \(p. 2377\)](#).

#### To stop and delete your human loop:

1. Once a human loop has been started, you can stop your human loop by calling the [StopHumanLoop](#) operation using the `HumanLoopName`. If a human loop was successfully stopped, the server sends back an HTTP 200 response.
2. To delete a human loop for which the status equals `Failed`, `Completed`, or `Stopped`, use the [DeleteHumanLoop](#) operation.

#### To list human loops:

1. You can list all active human loops by calling the [ListHumanLoops](#) operation. You can filter human loops by the creation date of the loop using the `CreationTimeAfter` and `CreateTimeBefore` parameters.
2. If successful, `ListHumanLoops` returns `HumanLoopSummaries` and `NextToken` objects in the response element. `HumanLoopSummaries` contains information about a single human loop. For example, it lists a loop's status and, if applicable, its failure reason.

Use the string returned in `NextToken` as an input in a subsequent call to `ListHumanLoops` to see the next page of human loops.

## Amazon A2I Output Data

When your machine learning workflow sends Amazon A2I a data object, a *human loop* is created and human reviewers receive a *task* to review that data object. The output data from each human review task is stored in the Amazon Simple Storage Service (Amazon S3) output bucket you specify in your human review workflow. In the path to the data, `YYYY/MM/DD/hh/mm/ss` represents the human loop creation date with year (`YYYY`), month (`MM`), and day (`DD`), and the creation time with hour (`hh`), minute (`mm`), and second (`ss`).

```
s3://customer-output-bucket-specified-in-flow-definition/flow-definition-name/YYYY/MM/DD/hh/mm/ss/human-loop-name/output.json
```

The content of your output data depends on the type of [task type](#) (built-in or custom) and the type of [workforce](#) you use. Your output data always includes the response from the human worker. Additionally, output data may include metadata about the human loop, the human reviewer (worker), and the data object.

Use the following sections to learn more about Amazon A2I output data format for different task types and workforces.

## Output Data From Built-In Task Types

Amazon A2I built-in task types include Amazon Textract and Amazon Rekognition. In addition to human responses, the output data from one of these tasks includes details about the reason the human loop was created and information about the integrated service used to create the human loop. Use the following table to learn more about the output data schema for all built-in task types. The *value* for each of these parameters depends on the service you use with Amazon A2I. Refer to the second table in this section for more information about these service-specific values.

Parameter	Value Type	Example Values	Description
awsManagedHumanLoopRequestSource	String	AWS/Rekognition/DetectModerationLabelsImage/V3 or AWS/Texttract/AnalyzeDocument/Forms/V1	The API operation and associated Amazon services that requested that Amazon A2I create the a human loop. This is the API operation you use to configure your Amazon A2I human loop.
flowDefinitionArn	String	arn:aws:sagemaker:us-west-2:111122223333:definition/flow-definition-name	The Amazon Resource Number (ARN) of the human review workflow (flow definition) used to create the human loop.
humanAnswers	List of JSON objects	<pre>{   "answerContent": {     "AWS/Rekognition/DetectModerationLabelsImage/V3": {       "moderationLabels": [         ...       }     },     ...   } }</pre> <p>or</p> <pre>{   "answerContent": {     "AWS/Texttract/AnalyzeDocument/Forms/V1": {       "blocks": [         ...       }     },     ...   } }</pre>	<p>A list of JSON objects that contain worker responses in <code>/answerContent</code>. This object also contains submission details and, if a private workforce was used, worker metadata. To learn more, see <a href="#">Track Worker Activity (p. 2385)</a>.</p> <p>For human loop output data produced from Amazon Rekognition DetectModerationLabel review tasks, this parameter only contains positive responses. For example, if workers select <i>No content</i>, this response is not included.</p>
humanLoopName	String	'human-loop-name'	The name of the human loop.
inputContent	JSON object	<pre>{   "aiServiceRequest": {...},   "aiServiceResponse": {...},   "humanTaskActivationConditionResults": {...} }</pre>	The input content the Amazon service sent to Amazon A2I when it requested a human loop be created.

Parameter	Value Type	Example Values	Description
		<pre>"selectedAiServiceResponse": { ... }</pre>	
aiServiceRequest	JSON object	<pre>{     "document": { ... },     "featureTypes": [ ... ],     "humanLoopConfig": { ... } }</pre> <p>or</p> <pre>{     "image": { ... },     "humanLoopConfig": { ... } }</pre>	The original request sent to the Amazon service integrated with Amazon A2I. For example, if you use Amazon Rekognition with Amazon A2I, this includes the request made through the API operation <code>DetectModerationLabels</code> . For Amazon Textract integrations, this includes the request made through <code>AnalyzeDocument</code> .
aiServiceResponse	JSON object	<pre>{     "moderationLabels": [ ... ],     "moderationModelVersion": "3.0" }</pre> <p>or</p> <pre>{     "blocks": [ ... ],     "documentMetadata": { } }</pre>	The full response from the Amazon service. This is the data that is used to determine if a human review is required. This object may contain metadata about the data object that is not shared with human reviewers.

Parameter	Value Type	Example Values	Description
selectedAiServiceResponse	JSON object	<pre>{     "moderationLabels": [...],     "moderationModelVersion": "3.0" }</pre> <p>or</p> <pre>{     "blocks": [...],     "documentMetadata": {} }</pre>	The subset of the aiServiceResponse that matches the activation conditions in ActivationConditions. All data objects listed in aiServiceResponse are listed in selectedAiServiceResponse when inferences are randomly sampled, or all inferences initiated activation conditions.
humanTaskActivationResults	JSON object	<pre>{     "Conditions": [...] }</pre>	A JSON object in inputContent that contains the reason a human loop was created. This includes a list of the activation conditions (Conditions) included in your human review workflow (flow definition), and the evaluation result for each condition—this result is either true or false. To learn more about activation conditions, see <a href="#">JSON Schema for Human Loop Activation Conditions in Amazon Augmented AI (p. 2345)</a> .

Select a tab on the following table to learn about the task type–specific parameters and see an example output-data code block for each of the built-in task types.

#### Amazon Textract Task Type Output Data

When you use the Amazon Textract built-in integration, you see 'AWS/Textract/AnalyzeDocument/Forms/V1' as the value for awsManagedHumanLoopRequestSource in your output data.

The answerContent parameter contains a Block object that includes human responses for all blocks sent to Amazon A2I.

The aiServiceResponse parameter also includes a Block object with Amazon Textract's response to the original request sent using to AnalyzeDocument.

To learn more about the parameters you see in the block object, refer to [Block](#) in the [Amazon Textract Developer Guide](#).

The following is an example of the output data from an Amazon A2I human review of Amazon Textract document analysis inferences.

```
{
    "awsManagedHumanLoopRequestSource": "AWS/Textract/AnalyzeDocument/Forms/V1",
    "flowDefinitionArn": "arn:aws:sagemaker:us-west-2:111122223333:flow-
definition/flow-definition-name",
    "humanAnswers": [
        {
            "answerContent": {
                "AWS/Textract/AnalyzeDocument/Forms/V1": {
                    "blocks": [...]
                }
            },
            "submissionTime": "2020-09-28T19:17:59.880Z",
            "workerId": "111122223333",
            "workerMetadata": {
                "identityData": {
                    "identityProviderType": "Cognito",
                    "issuer": "https://cognito-idp.us-west-2.amazonaws.com/us-
west-2_111111",
                    "sub": "c6aa8eb7-9944-42e9-a6b9-111122223333"
                }
            }
        }
    ],
    "humanLoopName": "humnan-loop-name",
    "inputContent": {
        "aiServiceRequest": {
            "document": {
                "s3Object": {
                    "bucket": "",
                    "name": "document-demo.jpg"
                }
            },
            "featureTypes": [
                "TABLES",
                "FORMS"
            ],
            "humanLoopConfig": {
                "dataAttributes": {
                    "contentClassifiers": [
                        "FreeOfPersonallyIdentifiableInformation"
                    ]
                },
                "flowDefinitionArn": "arn:aws:sagemaker:us-west-2:111122223333:flow-
definition/flow-definition-name",
                "humanLoopName": "humnan-loop-name"
            }
        },
        "aiServiceResponse": {
            "blocks": [...],
            "documentMetadata": {
                "pages": 1
            }
        },
        "humanTaskActivationConditionResults": {
            "Conditions": [
                {
                    "EvaluationResult": true,
                    "Or": [
                        {
                            "EvaluationResult": true
                        }
                    ]
                }
            ]
        }
    }
}
```

```
        "ConditionParameters": {
            "ImportantFormKey": "Mail address",
            "ImportantFormKeyAliases": [
                "Mail Address:",
                "Mail address:",
                "Mailing Add:",
                "Mailing Addresses"
            ],
            "KeyValueBlockConfidenceLessThan": 100,
            "WordBlockConfidenceLessThan": 100
        },
        "ConditionType": "ImportantFormKeyConfidenceCheck",
        "EvaluationResult": true
    },
    {
        "ConditionParameters": {
            "ImportantFormKey": "Mail address",
            "ImportantFormKeyAliases": [
                "Mail Address:",
                "Mail address:",
                "Mailing Add:",
                "Mailing Addresses"
            ]
        },
        "ConditionType": "MissingImportantFormKey",
        "EvaluationResult": false
    }
]
}
],
},
"selectedAiServiceResponse": {
    "blocks": [...]
}
}
```

## Amazon Rekognition Task Type Output Data

When you use the Amazon Textract built-in integration, you see the string 'AWS/Rekognition/DetectModerationLabels/Image/V3' as the value for `awsManagedHumanLoopRequestSource` in your output data.

The `answerContent` parameter contains a `moderationLabels` object that contains human responses for all moderation labels sent to Amazon A2I.

The `aiServiceResponse` parameter also includes a `moderationLabels` object with Amazon Rekognition's response to the original request sent to `DetectModerationLabels`.

To learn more about the parameters you see in the block object, refer to [ModerationLabel](#) in the Amazon Rekognition Developer Guide.

The following is an example of the output data from an Amazon A2I human review of Amazon Rekognition image moderation inferences.

```
V3",
    "flowDefinitionArn": "arn:aws:sagemaker:us-west-2:111122223333:flow-
definition/flow-definition-name",
    "humanAnswers": [
        {
            "answerContent": {
```

```

    "AWS/Rekognition/DetectModerationLabels/Image/V3": {
        "moderationLabels": [...]
    }
},
"submissionTime": "2020-09-28T19:22:35.508Z",
"workerId": "ef7294f850a3d9d1",
"workerMetadata": {
    "identityData": {
        "identityProviderType": "Cognito",
        "issuer": "https://cognito-idp.us-west-2.amazonaws.com/us-
west-2_111111",
        "sub": "c6aa8eb7-9944-42e9-a6b9-111122223333"
    }
}
],
"humanLoopName": "humnan-loop-name",
"inputContent": {
    "aiServiceRequest": {
        "humanLoopConfig": {
            "flowDefinitionArn": "arn:aws:sagemaker:us-west-2:111122223333:flow-
definition/flow-definition-name",
            "humanLoopName": "humnan-loop-name"
        },
        "image": {
            "s3Object": {
                "bucket": "",
                "name": "example-image.jpg"
            }
        }
    },
    "aiServiceResponse": {
        "moderationLabels": [...],
        "moderationModelVersion": "3.0"
    },
    "humanTaskActivationConditionResults": {
        "Conditions": [
            {
                "EvaluationResult": true,
                "Or": [
                    {
                        "ConditionParameters": {
                            "ConfidenceLessThan": 98,
                            "ModerationLabelName": "Suggestive"
                        },
                        "ConditionType": "ModerationLabelConfidenceCheck",
                        "EvaluationResult": true
                    },
                    {
                        "ConditionParameters": {
                            "ConfidenceGreaterThan": 98,
                            "ModerationLabelName": "Female Swimwear Or Underwear"
                        },
                        "ConditionType": "ModerationLabelConfidenceCheck",
                        "EvaluationResult": false
                    }
                ]
            }
        ],
        "selectedAiServiceResponse": {
            "moderationLabels": [
                {
                    "confidence": 96.7122802734375,
                    "name": "Suggestive",
                    "parentName": ""
                }
            ]
        }
    }
}

```

```

        }
    ],
    "moderationModelVersion": "3.0"
}
}
}
```

## Output Data From Custom Task Types

When you add Amazon A2I to a custom human review workflow, you see the following parameters in the output data returned from human review tasks.

Parameter	Value Type	Description
<code>flowDefinitionArn</code>	String	The Amazon Resource Number (ARN) of the human review workflow (flow definition) used to create the human loop.
<code>humanAnswers</code>	List of JSON objects	A list of JSON objects that contain worker responses in <code>answerContent</code> . The value in this parameter is determined by the output received from your <a href="#">worker task template</a> .  If you are using a private workforce, worker metadata is included. To learn more, see <a href="#">Track Worker Activity (p. 2385)</a> .
<code>humanLoopName</code>	String	The name of the human loop.
<code>inputContent</code>	JSON Object	The input content sent to Amazon A2I in the request to <a href="#">StartHumanLoop</a> .

The following is an example of output data from a custom integration with Amazon A2I and Amazon Transcribe. In this example, the `inputContent` consists of:

- A path to an .mp4 file in Amazon S3 and the video title
- The transcription returned from Amazon Transcribe (parsed from Amazon Transcribe output data)
- A start and end time used by the worker task template to clip the .mp4 file and show workers a relevant portion of the video

```

{
    "flowDefinitionArn": "arn:aws:sagemaker:us-west-2:111122223333:flow-definition/flow-definition-name",
    "humanAnswers": [
        {
            "answerContent": {
                "transcription": "use lambda to turn your notebook"
            },
            "submissionTime": "2020-06-18T17:08:26.246Z",
            "workerId": "ef7294f850a3d9d1",

```

```

        "workerMetadata": {
            "identityData": {
                "identityProviderType": "Cognito",
                "issuer": "https://cognito-idp.us-west-2.amazonaws.com/us-
west-2_111111",
                "sub": "c6aa8eb7-9944-42e9-a6b9-111122223333"
            }
        }

    ],
    "humanLoopName": "human-loop-name",
    "inputContent": {
        "audioPath": "s3://a2i_transcribe_demo/Fully-Managed Notebook Instances with
Amazon SageMaker - a Deep Dive.mp4",
        "end_time": 950.27,
        "original_words": "but definitely use Lambda to turn your ",
        "start_time": 948.51,
        "video_title": "Fully-Managed Notebook Instances with Amazon SageMaker - a Deep
Dive.mp4"
    }
}

```

## Track Worker Activity

Amazon A2I provides information that you can use to track individual workers in task output data. To identify the worker that worked on the human review task, use the following from the output data in Amazon S3:

- The `acceptanceTime` is the time that the worker accepted the task. The format of this date and time stamp is `YYYY-MM-DDTHH:MM:SS.mmmZ` for the year (`YYYY`), month (`MM`), day (`DD`), hour (`HH`), minute (`MM`), second (`SS`), and millisecond (`mmm`). The date and time are separated by a `T`.
- The `submissionTime` is the time that the worker submitted their annotations using the **Submit** button. The format of this date and time stamp is `YYYY-MM-DDTHH:MM:SS.mmmZ` for the year (`YYYY`), month (`MM`), day (`DD`), hour (`HH`), minute (`MM`), second (`SS`), and millisecond (`mmm`). The date and time are separated by a `T`.
- `timeSpentInSeconds` reports the total time, in seconds, that a worker actively worked on that task. This metric does not include time when a worker paused or took a break.
- The `workerId` is unique to each worker.
- If you use a [private workforce](#), in `workerMetadata`, you see the following.
  - The `identityProviderType` is the service used to manage the private workforce.
  - The `issuer` is the Amazon Cognito user pool or OpenID Connect (OIDC) Identity Provider (IdP) issuer associated with the work team assigned to this human review task.
  - A unique `sub` identifier refers to the worker. If you create a workforce using Amazon Cognito, you can retrieve details about this worker (such as the name or user name) associated with this ID using Amazon Cognito. To learn how, see [Managing and Searching for User Accounts in Amazon Cognito Developer Guide](#).

The following is an example of the output you may see if you use Amazon Cognito to create a private workforce. This is identified in the `identityProviderType`.

```

"submissionTime": "2020-12-28T18:59:58.321Z",
"acceptanceTime": "2020-12-28T18:59:15.191Z",
"timeSpentInSeconds": 40.543,
"workerId": "a12b3cdefg4h5i67",
"workerMetadata": {
    "identityData": {

```

```
        "identityProviderType": "Cognito",
        "issuer": "https://cognito-idp..amazonaws.com/aws-region_123456789",
        "sub": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee"
    }
}
```

The following is an example of the output you may see if you use your own OIDC IdP to create a private workforce:

```
"workerMetadata": {
    "identityData": {
        "identityProviderType": "Oidc",
        "issuer": "https://example-oidc-ipd.com/adfs",
        "sub": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee"
    }
}
```

To learn more about using private workforces, see [Use a Private Workforce \(p. 461\)](#).

## Permissions and Security in Amazon Augmented AI

When using Amazon Augmented AI (Amazon A2I) to create a human review workflow for your ML/AI application, you create and configure *resources* in Amazon SageMaker such as a human workforce and worker task templates. To configure and start a human loop, you either integrate Amazon A2I with other Amazon services such as Amazon Textract or Amazon Rekognition, or use the Amazon Augmented AI Runtime API. To create a human review workflow and start a human loop, you must attach certain policies to your Amazon Identity and Access Management (IAM) role or user. Specifically:

- When you start a human loop using image input data on or after January 12th, 2020, you must add a CORS header policy to the Amazon S3 bucket that contains your input data. See [CORS Permission Requirement \(p. 2387\)](#) to learn more.
- When you create a flow definition, you need to provide a role that grants Amazon A2I permission to access Amazon S3 both for reading objects that are rendered in a human task UI and for writing the results of the human review.

This role must also have a trust policy attached to give SageMaker permission to assume the role. This allows Amazon A2I to perform actions in accordance with permissions that you attach to the role.

See [Add Permissions to the IAM Role Used to Create a Flow Definition \(p. 2388\)](#) for example policies that you can modify and attach to the role you use to create a flow definition. These are the policies that are attached to the IAM role that is created in the **Human review workflows** section of the Amazon A2I area of the SageMaker console.

- To create and start human loops, you either use an API operation from a built-in task type (such as `DetectModerationLabel` or `AnalyzeDocument`) or the Amazon A2I Runtime API operation `StartHumanLoop` in a custom ML application. You need to attach the `AmazonAugmentedAIFullAccess` managed policy to the IAM user that invokes these API operations to grant permission to these services to use Amazon A2I operations. To learn how, see [Create an IAM User That Can Invoke Amazon A2I API Operations \(p. 2389\)](#).

This policy does *not* grant permission to invoke the API operations of the Amazon service associated with built-in task types. For example, `AmazonAugmentedAIFullAccess` does not grant permission to call the Amazon Rekognition `DetectModerationLabel` API operation or Amazon Textract `AnalyzeDocument` API operation. You can use the more general policy, `AmazonAugmentedAIIntegratedAPIAccess`, to grant these permissions. For more information, see [Create an IAM User With Permissions to Invoke Amazon A2I, Amazon Textract, and Amazon](#)

[Rekognition API Operations \(p. 2390\)](#). This is a good option when you want to grant an IAM user broad permissions to use Amazon A2I and integrated Amazon services' API operations.

If you want to configure more granular permissions, see [Amazon Rekognition Identity-Based Policy Examples](#) and [Amazon Textract Identity-Based Policy Examples](#) for identity-based policies you can use to grant permission to use these individual services.

- To preview your custom worker task UI template, you need an IAM role with permissions to read Amazon S3 objects that get rendered on your user interface. See a policy example in [Enable Worker Task Template Previews \(p. 2390\)](#).

#### Topics

- [CORS Permission Requirement \(p. 2387\)](#)
- [Add Permissions to the IAM Role Used to Create a Flow Definition \(p. 2388\)](#)
- [Create an IAM User That Can Invoke Amazon A2I API Operations \(p. 2389\)](#)
- [Create an IAM User With Permissions to Invoke Amazon A2I, Amazon Textract, and Amazon Rekognition API Operations \(p. 2390\)](#)
- [Enable Worker Task Template Previews \(p. 2390\)](#)
- [Using Amazon A2I with Amazon KMS Encrypted Buckets \(p. 2391\)](#)
- [Additional Permissions and Security Resources \(p. 2392\)](#)

## CORS Permission Requirement

Earlier in 2020, widely used browsers like Chrome and Firefox changed their default behavior for rotating images based on image metadata, referred to as [EXIF data](#). Previously, images would always display in browsers exactly how they are stored on disk, which is typically unrotated. After the change, images now rotate according to a piece of image metadata called *orientation value*. This has important implications for the entire machine learning (ML) community. For example, if the EXIF orientation is not considered, applications that are used to annotate images may display images in unexpected orientations and result in incorrect labels.

Starting with Chrome 89, Amazon can no longer automatically prevent the rotation of images because the web standards group W3C has decided that the ability to control rotation of images violates the web's Same-Origin Policy. Therefore, to ensure human workers annotate your input images in a predictable orientation when you submit requests to create a human loop, you must add a CORS header policy to the S3 buckets that contain your input images.

#### Important

If you do not add a CORS configuration to the S3 buckets that contains your input data, human review tasks for those input data objects fail.

You can add a CORS policy to an S3 bucket that contains input data in the Amazon S3 console. To set the required CORS headers on the S3 bucket that contains your input images in the S3 console, follow the directions detailed in [How do I add cross-domain resource sharing with CORS?](#). Use the following CORS configuration code for the buckets that host your images. If you use the Amazon S3 console to add the policy to your bucket, you must use the JSON format.

#### JSON

```
[  
  {  
    "AllowedHeaders": [],  
    "AllowedMethods": ["GET"],  
    "AllowedOrigins": ["*"],  
    "ExposeHeaders": []  
  }]
```

#### XML

```
<CORSConfiguration>
  <CORSRule>
    <AllowedOrigin>*</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
  </CORSRule>
</CORSConfiguration>
```

## Add Permissions to the IAM Role Used to Create a Flow Definition

To create a flow definition, attach the policies in this section to the role that you use when creating a human review workflow in the SageMaker console, or when using the `CreateFlowDefinition` API operation.

- If you are using the console to create a human review workflow, enter the role Amazon Resource Name (ARN) in the **IAM role** field when [creating a human review workflow in the console](#).
- When creating a flow definition using the API, attach these policies to the role that is passed to the `RoleArn` parameter of the `CreateFlowDefinition` operation.

When you create a human review workflow (flow definition), Amazon A2I invokes Amazon S3 to complete your task. To grant Amazon A2I permission to retrieve and store your files in your Amazon S3 bucket, create the following policy and attach it to your role. For example, if the images, documents, and other files that you are sending for human review are stored in an S3 bucket named `my_input_bucket`, and if you want the human reviews to be stored in a bucket named `my_output_bucket`, create the following policy.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::my_input_bucket/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::my_output_bucket/*"
      ]
    }
  ]
}
```

In addition, the IAM role must have the following trust policy to give SageMaker permission to assume the role. To learn more about IAM trust policies, see [Resource-Based Policies](#) section of **Policies and Permissions** in the *Amazon Identity and Access Management* documentation.

```
{
```

```
"Version": "2012-10-17",
"Statement": [
  {
    "Sid": "AllowSageMakerToAssumeRole",
    "Effect": "Allow",
    "Principal": {
      "Service": "sagemaker.amazonaws.com"
    },
    "Action": "sts:AssumeRole"
  }
]
```

For more information about creating and managing IAM roles and policies, see the following topics in the *Amazon Identity and Access Management User Guide*:

- To create an IAM role, see [Creating a Role to Delegate Permissions to an IAM User](#).
- To learn how to create IAM policies, see [Creating IAM Policies](#).
- To learn how to attach an IAM policy to a role, see [Adding and Removing IAM Identity Permissions](#).

## Create an IAM User That Can Invoke Amazon A2I API Operations

To use Amazon A2I to create and start human loops for Amazon Rekognition, Amazon Textract, or the Amazon A2I runtime API, you must use an IAM user that has permissions to invoke Amazon A2I operations. To do this, use the IAM console to attach the [AmazonAugmentedAIFullAccess](#) managed policy to a new or existing IAM user.

This policy grants permission to an IAM user to invoke API operations from the SageMaker API for flow definition creation and management and the Amazon Augmented AI Runtime API for human loop creation and management. To learn more about these API operations, see [Use APIs in Amazon Augmented AI](#).

[AmazonAugmentedAIFullAccess](#) does not grant permissions to use Amazon Rekognition or Amazon Textract API operations.

### Note

You can also attach the [AmazonAugmentedAIFullAccess](#) policy to an IAM role that is used to create and start a human loop.

### To create the required IAM user

1. Sign in to the IAM console at <https://console.amazonaws.cn/iam/>.
2. Choose **Users** and choose an existing user, or create a new user by choosing **Add user**. To learn how to create a new user, see [Creating an IAM User in Your Amazon Account](#) in the *Amazon Identity and Access Management User Guide*.
  - If you chose to attach the policy to an existing user, choose **Add permissions**.
  - While creating a new user, follow the next step on the **Set permissions** page.
3. Choose **Attach existing policies directly**.
4. In the **Search** bar, enter [AmazonAugmentedAIFullAccess](#) and check the box next to that policy.

To enable this IAM user to create a flow definition with the public work team, also attach the [AmazonSageMakerMechanicalTurkAccess](#) managed policy.
5. After attaching the policy or policies:

- a. If you are using an existing user, choose **Next: Review**, and then choose **Add permissions**.
- b. If you are creating a new user, choose **Next: Tags** and complete the process of creating your user.

For more information, see [Adding and Removing IAM Identity Permissions](#) in the *Amazon Identity and Access Management User Guide*.

## Create an IAM User With Permissions to Invoke Amazon A2I, Amazon Textract, and Amazon Rekognition API Operations

To create an IAM user that has permission to invoke the API operations used by the built-in task types (that is, `DetectModerationLabels` for Amazon Rekognition and `AnalyzeDocument` for Amazon Textract) and permission to use all Amazon A2I API operations, attach the IAM managed policy, `AmazonAugmentedAIIIntegratedAPIAccess`. You may want to use this policy when you want to grant broad permissions to an IAM user using Amazon A2I with more than one task type. To learn more about these API operations, see [Use APIs in Amazon Augmented AI](#).

**Note**

You can also attach the `AmazonAugmentedAIIIntegratedAPIAccess` policy to an IAM role that is used to create and start a human loop.

### To create the required IAM user

1. Sign in to the IAM console at <https://console.amazonaws.cn/iam/>.
2. Choose **Users** and choose an existing user, or create a new user by choosing **Add user**. To learn how to create a new user, see [Creating an IAM User in Your Amazon Account](#) in the *Amazon Identity and Access Management User Guide*.

- If you chose to attach the policy to an existing user, choose **Add permissions**.
- If you are creating a new user, follow the next step on the **Set permissions** page.

3. Choose **Attach existing policies directly**.

4. In the **Search** bar, enter `AmazonAugmentedAIIIntegratedAPIAccess` and select the box next to that policy.

To allow this IAM user to create a flow definition using Amazon Mechanical Turk, also attach the `AmazonSageMakerMechanicalTurkAccess` managed policy.

5. After attaching the policy or policies:

- a. If you are using an existing user, choose **Next: Review**, and then choose **Add permissions**.
- b. If you are creating a new user, choose **Next: Tags** and complete the process of creating your user.

For more information, see [Adding and Removing IAM Identity Permissions](#) in the *Amazon Identity and Access Management User Guide*.

## Enable Worker Task Template Previews

To customize the interface and instructions that your workers see when working on your tasks, you create a worker task template. You can create the template using the `CreateHumanTaskUi` operation or the SageMaker console.

To preview your template, you need an IAM role with the following permissions to read Amazon S3 objects that get rendered on your user interface.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::my_input_bucket/*"
            ]
        }
    ]
}
```

For Amazon Rekognition and Amazon Textract task types, you can preview your template using the Amazon Augmented AI section of the SageMaker console. For custom task types, you preview your template by invoking the [RenderUiTemplate](#) operation. To preview your template, follow the instructions for your task type:

- Amazon Rekognition and Amazon Textract task types – In the SageMaker console, use the role's Amazon Resource Name (ARN) in the procedure documented in [Create a Worker Task Template \(p. 2367\)](#).
- Custom task types – In the `RenderUiTemplate` operation, use the role's ARN in the `RoleArn` parameter.

## Using Amazon A2I with Amazon KMS Encrypted Buckets

If you specify an Amazon KMS customer managed CMK to encrypt output data in `outputConfig` of [CreateFlowDefinition](#), you must add an IAM policy similar to the following to that key. This policy gives the IAM execution role that you use to create your human loops permission to use this key to perform all of the actions listed in "Action". To learn more about these actions, see [Amazon KMS permissions](#) in the Amazon Key Management Service Developer Guide.

To use this policy, replace the IAM service-role ARN in "Principal" with the ARN of the execution role you use to create the human review workflow (flow definition). When you create a labeling job using `CreateFlowDefinition`, this is the ARN you specify for `RoleArn`. Note that you cannot provide a `KmsKeyId` when you create a human loop in the console.

```
{
    "Sid": "AllowUseOfKmsKey",
    "Effect": "Allow",
    "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/service-role/example-role"
    },
    "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey"
    ],
    "Resource": "*"
}
```

}

## Additional Permissions and Security Resources

- the section called “Control Access to SageMaker Resources by Using Tags” (p. 2444).
- the section called “SageMaker Identity-Based Policies” (p. 2421)
- the section called “Control Creation of SageMaker Resources with Condition Keys” (p. 2434)
- the section called “Amazon SageMaker API Permissions Reference” (p. 2474)
- *Security* (p. 2410)

## Use Amazon CloudWatch Events in Amazon Augmented AI

Amazon Augmented AI uses Amazon CloudWatch Events to alert you when a human review loop status changes to Completed, Failed, or Stopped. This event delivery is guaranteed at least once, which means all events created when human loops finish are successfully delivered to CloudWatch Events (Amazon EventBridge). When a review loop changes to one of these states, Augmented AI sends an event to CloudWatch Events similar to the following.

```
{  
    "version": "0",  
    "id": "12345678-1111-2222-3333-12345EXAMPLE",  
    "detail-type": "SageMaker A2I HumanLoop Status Change",  
    "source": "aws.sagemaker",  
    "account": "111111111111",  
    "time": "2019-11-14T17:49:25Z",  
    "region": "us-east-1",  
    "resources": ["arn:aws:sagemaker:us-east-1:111111111111:human-loop/humanloop-nov-14-1"],  
    "detail": {  
        "creationTime": "2019-11-14T17:37:36.740Z",  
        "failureCode": null,  
        "failureReason": null,  
        "flowDefinitionArn": "arn:aws:sagemaker:us-east-1:111111111111:flow-definition/  
flowdef-nov-12",  
        "humanLoopArn": "arn:aws:sagemaker:us-east-1:111111111111:human-loop/humanloop-  
nov-14-1",  
        "humanLoopName": "humanloop-nov-14-1",  
        "humanLoopOutput": {  
            "outputS3Uri": "s3://customer-output-bucket-specified-in-flow-definition/  
flowdef-nov-12/2019/11/14/17/37/36/humanloop-nov-14-1/output.json"  
        },  
        "humanLoopStatus": "Completed"  
    }  
}
```

The details in the JSON output include the following:

**creationTime**

The timestamp when Augmented AI created the human loop.

**failureCode**

A failure code denoting a specific type of failure.

**failureReason**

The reason why a human loop has failed. The failure reason is only returned when the human review loop status is failed.

**flowDefinitionArn**

The Amazon Resource Name (ARN) of the flow definition, or *human review workflow*.

**humanLoopArn**

The Amazon Resource Name (ARN) of the human loop.

**humanLoopName**

The name of the human loop.

**humanLoopOutput**

An object containing information about the output of the human loop.

**outputS3Uri**

The location of the Amazon S3 object where Augmented AI stores your human loop output.

**humanLoopStatus**

The status of the human loop.

## Send Events from Your Human Loop to CloudWatch Events

To configure a CloudWatch Events rule to get status updates, or *events*, for your Amazon A2I human loops, use the Amazon Command Line Interface (Amazon CLI) [put-rule](#) command. When using the [put-rule](#) command, specify the following to receive human loop statuses:

- `\\"source\\":[\\\"aws.sagemaker\\\"]`
- `\\"detail-type\\":[\\\"SageMaker A2I HumanLoop Status Change\\\"]`

To configure a CloudWatch Events rule to watch for all status changes, use the following command and replace the placeholder text. For example, replace [`"A2IHumanLoopStatusChanges"`](#) with a unique CloudWatch Events rule name and [`"arn:aws:iam::111122223333:role/MyRoleForThisRule"`](#) with the Amazon Resource Number (ARN) of an IAM role with an `events.amazonaws.com` trust policy attached. Replace `region` with the Amazon Region in which you want to create the rule.

```
aws events put-rule --name "A2IHumanLoopStatusChanges"
  --event-pattern "\\"source\\":[\\\"aws.sagemaker\\\"], \\"detail-type\\":[\\\"SageMaker A2I HumanLoop Status Change\\\"]"
  --role-arn "arn:aws:iam::111122223333:role/MyRoleForThisRule"
  --region "region"
```

To learn more about the `put-rule` request, see [Event Patterns in CloudWatch Events](#) in the *Amazon CloudWatch Events User Guide*.

## Set Up a Target to Process Events

To process events, you need to set up a target. For example, if you want to receive an email when a human loop status changes, use a procedure in [Setting Up Amazon SNS Notifications](#) in the *Amazon*

*CloudWatch User Guide* to set up an Amazon SNS topic and subscribe your email to it. Once you have created a topic, you can use it to create a target.

### To add a target to your CloudWatch Events rule

1. Open the CloudWatch console: <https://console.amazonaws.cn/cloudwatch/home>
2. In the navigation pane, choose **Rules**.
3. Choose the rule to which you want to add a target.
4. Choose **Actions**, and then choose **Edit**.
5. Under **Targets**, choose **Add Target** and choose the Amazon service you want to act when a human loop status change event is detected.
6. Configure your target. For instructions, see the topic for configuring a target in the [Amazon documentation for that service](#).
7. Choose **Configure details**.
8. For **Name**, enter a name and, optionally, provide details about the purpose of the rule in **Description**.
9. Make sure that the check box next to **State** is selected so that your rule is listed as **Enabled**.
10. Choose **Update rule**.

## Use Human Review Output

After you receive human review results, you can analyze the results and compare them to machine learning predictions. The JSON that is stored in the Amazon S3 bucket contains both the machine learning predictions and the human review results.

## More Information

[Automating Amazon SageMaker with Amazon EventBridge \(p. 2537\)](#)

## Use APIs in Amazon Augmented AI

You can create a human review workflow or a worker task template programmatically. The APIs you use depend on whether you are creating a Amazon Rekognition, Amazon Textract, or custom task type. This topic provides links to API reference documentation for each task type and programming task.

The following APIs can be used with Augmented AI:

### Amazon Augmented AI

Use the Augmented AI API to start, stop, and delete human review loops. You can also list all human review loops and return information about human review loops in your account.

Learn more about human review loop APIs in the [Amazon Augmented AI Runtime API Reference](#).

### Amazon Rekognition

Use the **HumanLoopConfig** parameter of the [`DetectModerationLabels`](#) API to initiate a human review workflow using Amazon Rekognition.

### Amazon SageMaker

Use the Amazon SageMaker API to create a `FlowDefinition`, also known as a *human review workflow*. You can also create a `HumanTaskUi` or `worker task template`.

For more information, see the [CreateFlowDefinition](#) or the [CreateHumanTaskUi](#) API documentation.

#### Amazon Textract

Use the **HumanLoopConfig** parameter of the [AnalyzeDocument](#) API to initiate a human review workflow using Amazon Textract.

## Programmatic Tutorials

The following tutorials provide example code and step-by-step instructions for creating human review workflows and worker task templates programmatically.

- [Tutorial: Get Started Using the Amazon A2I API \(p. 2320\)](#)
- [Create a Human Review Workflow \(API\) \(p. 2341\)](#)
- [Create and Start a Human Loop \(p. 2358\)](#)
- [Using Amazon Augmented AI with Amazon Rekognition in the \*Amazon Rekognition Developer Guide\*](#)
- [Using Amazon Augmented AI with Amazon Textract AnalyzeDocument in the \*Amazon Textract Developer Guide\*](#)

# Buy and Sell Amazon SageMaker Algorithms and Models in Amazon Web Services Marketplace

This feature is not available in the China Regions.

Amazon SageMaker integrates with Amazon Web Services Marketplace, enabling developers to charge other SageMaker users for the use of their algorithms and model packages. Amazon Web Services Marketplace is a curated digital catalog that makes it easy for customers to find, buy, deploy, and manage third-party software and services that customers need to build solutions and run their businesses. Amazon Web Services Marketplace includes thousands of software listings in popular categories, such as security, networking, storage, machine learning, business intelligence, database, and DevOps. It simplifies software licensing and procurement with flexible pricing options and multiple deployment methods.

## Use your own algorithms and models with the Amazon Marketplace

The following sections show how to create algorithm and model package resources that you can use locally and publish to the Amazon Marketplace.

### Topics

- [Create Algorithm and Model Package Resources \(p. 2396\)](#)
- [Use Algorithm and Model Package Resources \(p. 2402\)](#)

## Create Algorithm and Model Package Resources

After your training and/or inference code is packaged in Docker containers, create algorithm and model package resources that you can use in your Amazon SageMaker account and, optionally, publish on Amazon Web Services Marketplace.

### Topics

- [Create an Algorithm Resource \(p. 2396\)](#)
- [Create a Model Package Resource \(p. 2400\)](#)

## Create an Algorithm Resource

To create an algorithm resource that you can use to run training jobs in Amazon SageMaker and publish on Amazon Web Services Marketplace specify the following information:

- The Docker containers that contains the training and, optionally, inference code.
- The configuration of the input data that your algorithm expects for training.
- The hyperparameters that your algorithm supports.

- Metrics that your algorithm sends to Amazon CloudWatch during training jobs.
- The instance types that your algorithm supports for training and inference, and whether it supports distributed training across multiple instances.
- Validation profiles, which are training jobs that SageMaker uses to test your algorithm's training code and batch transform jobs that SageMaker runs to test your algorithm's inference code.

To ensure that buyers and sellers can be confident that products work in SageMaker, we require that you validate your algorithms before listing them on Amazon Web Services Marketplace. You can list products in the Amazon Web Services Marketplace only if validation succeeds. To validate your algorithms, SageMaker uses your validation profile and sample data to run the following validations tasks:

1. Create a training job in your account to verify that your training image works with SageMaker.
2. If you included inference code in your algorithm, create a model in your account using the algorithm's inference image and the model artifacts produced by the training job.
3. If you included inference code in your algorithm, create a transform job in your account using the model to verify that your inference image works with SageMaker.

When you list your product on Amazon Web Services Marketplace, the inputs and outputs of this validation process persist as part of your product and are made available to your buyers. This helps buyers understand and evaluate the product before they buy it. For example, buyers can inspect the input data that you used, the outputs generated, and the logs and metrics emitted by your code. The more comprehensive your validation specification, the easier it is for customers to evaluate your product.

**Note**

In your validation profile, provide only data that you want to expose publicly.

Validation can take up to a few hours. To see the status of the jobs in your account, in the SageMaker console, see the **Training jobs** and **Transform jobs** pages. If validation fails, you can access the scan and validation reports from the SageMaker console. If any issues are found, you will have to create the algorithm again.

**Note**

To publish your algorithm on Amazon Web Services Marketplace, at least one validation profile is required.

You can create an algorithm by using either the SageMaker console or the SageMaker API.

**Topics**

- [Create an Algorithm Resource \(Console\) \(p. 2397\)](#)
- [Create an Algorithm Resource \(API\) \(p. 2400\)](#)

## Create an Algorithm Resource (Console)

### To create an algorithm resource (console)

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Choose **Algorithms**, then choose **Create algorithm**.
3. On the **Training specifications** page, provide the following information:
  - a. For **Algorithm name**, type a name for your algorithm. The algorithm name must be unique in your account and in the Amazon region. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
  - b. Type a description for your algorithm. This description appears in the SageMaker console and in the Amazon Web Services Marketplace.

- c. For **Training image**, type the path in Amazon ECR where your training container is stored.
  - d. For **Support distributed training**, Choose Yes if your algorithm supports training on multiple instances. Otherwise, choose No.
  - e. For **Support instance types for training**, choose the instance types that your algorithm supports.
  - f. For **Channel specification**, specify up to 8 channels of input data for your algorithm. For example, you might specify 3 input channels named train, validation, and test. For each channel, specify the following information:
    - i. For **Channel name**, type a name for the channel. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
    - ii. To require the channel for your algorithm, choose **Channel required**.
    - iii. Type a description for the channel.
    - iv. For **Supported input modes**, choose **Pipe mode** if your algorithm supports streaming the input data, and **File mode** if your algorithm supports downloading the input data as a file. You can choose both.
    - v. For **Supported content types**, type the MIME type that your algorithm expects for input data.
    - vi. For **Supported compression type**, choose **Gzip** if your algorithm supports Gzip compression. Otherwise, choose **None**.
    - vii. Choose **Add channel** to add another data input channel, or choose **Next** if you are done adding channels.
4. On the **Tuning specifications** page, provide the following information:

- a. For **Hyperparameter specification**, specify the hyperparameters that your algorithm supports by editing the JSON object. For each hyperparameter that your algorithm supports, construct a JSON block similar to the following:

```
{
  "DefaultValue": "5",
  "Description": "The first hyperparameter",
  "IsRequired": true,
  "IsTunable": false,
  "Name": "intRange",
  "Range": {
    "IntegerParameterRangeSpecification": {
      "MaxValue": "10",
      "MinValue": "1"
    },
    "Type": "Integer"
  }
}
```

In the JSON, supply the following:

- i. For **DefaultValue**, specify a default value for the hyperparameter, if there is one.
- ii. For **Description**, specify a description for the hyperparameter.
- iii. For **IsRequired**, specify whether the hyperparameter is required.
- iv. For **IsTunable**, specify **true** if this hyperparameter can be tuned when a user runs a hyperparameter tuning job that uses this algorithm. For information, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).
- v. For **Name**, specify a name for the hyperparameter.
- vi. For **Range**, specify one of the following:
  - **IntegerParameterRangeSpecification** - the values of the hyperparameter are integers. Specify minimum and maximum values for the hyperparameter.

- - **ContinuousParameterRangeSpecification** - the values of the hyperparameter are floating-point values. Specify minimum and maximum values for the hyperparameter.
    - **CategoricalParameterRangeSpecification** - the values of the hyperparameter are categorical values. Specify a list of all of the possible values.
- vii. For **Type**, specify **Integer**, **Continuous**, or **Categorical**. The value must correspond to the type of Range that you specified.
- b. For **Metric definitions**, specify any training metrics that you want your algorithm to emit. SageMaker uses the regular expression that you specify to find the metrics by parsing the logs from your training container during training. Users can view these metrics when they run training jobs with your algorithm, and they can monitor and plot the metrics in Amazon CloudWatch. For information, see [Monitor and Analyze Training Jobs Using Metrics \(p. 1868\)](#). For each metric, provide the following information:
- i. For **Metric name**, type a name for the metric.
  - ii. For **Regex**, type the regular expression that SageMaker uses to parse training logs so that it can find the metric value.
  - iii. For **Objective metric support** choose **Yes** if this metric can be used as the objective metric for a hyperparameter tuning job. For information, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).
  - iv. Choose **Add metric** to add another metric, or choose **Next** if you are done adding metrics.
5. On the **Inference specifications** page, provide the following information if your algorithm supports inference:
- a. For **Container definition**, type path in Amazon ECR where your inference container is stored.
  - b. For **Container DNS host name**, type the name of a DNS host for your image.
  - c. For **Supported instance types for real-time inference**, choose the instance types that your algorithm supports for models deployed as hosted endpoints in SageMaker. For information, see [Deploy a Model on SageMaker Hosting Services \(p. 10\)](#).
  - d. For **Supported instance types for batch transform jobs**, choose the instance types that your algorithm supports for batch transform jobs. For information, see [Get Inferences for an Entire Dataset with Batch Transform \(p. 13\)](#).
  - e. For **Supported content types**, type the type of input data that your algorithm expects for inference requests.
  - f. For **Supported response MIME types**, type the MIME types that your algorithm supports for inference responses.
  - g. Choose **Next**.
6. On the **Validation specifications** page, provide the following information:
- a. For **Publish this algorithm on Amazon Web Services Marketplace**, choose **Yes** to publish the algorithm on Amazon Web Services Marketplace.
  - b. For **Validate this algorithm**, choose **Yes** if you want SageMaker to run training jobs and/or batch transform jobs that you specify to test the training and/or inference code of your algorithm.

#### Note

To publish your algorithm on Amazon Web Services Marketplace, your algorithm must be validated.

- c. For **IAM role**, choose an IAM role that has the required permissions to run training jobs and batch transform jobs in SageMaker, or choose **Create a new role** to allow SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see [SageMaker Roles \(p. 2447\)](#).
- d. For **Validation profile**, specify the following:

- A name for the validation profile.
  - A **Training job definition**. This is a JSON block that describes a training job. This is in the same format as the `TrainingJobDefinition` input parameter of the `CreateAlgorithm` API.
  - A **Transform job definition**. This is a JSON block that describes a batch transform job. This is in the same format as the `TransformJobDefinition` input parameter of the `CreateAlgorithm` API.
- e. Choose **Create algorithm**.

## Create an Algorithm Resource (API)

To create an algorithm resource by using the SageMaker API, call the `CreateAlgorithm` API.

## Create a Model Package Resource

To create a model package resource that you can use to create deployable models in Amazon SageMaker and publish on Amazon Web Services Marketplace specify the following information:

- The Docker container that contains the inference code, or the algorithm resource that was used to train the model.
- The location of the model artifacts. Model artifacts can either be packaged in the same Docker container as the inference code or stored in Amazon S3.
- The instance types that your model package supports for both real-time inference and batch transform jobs.
- Validation profiles, which are batch transform jobs that SageMaker runs to test your model package's inference code.

Before listing model packages on Amazon Web Services Marketplace, you must validate them. This ensures that buyers and sellers can be confident that products work in Amazon SageMaker. You can list products on Amazon Web Services Marketplace only if validation succeeds.

The validation procedure uses your validation profile and sample data to run the following validations tasks:

1. Create a model in your account using the model package's inference image and the optional model artifacts that are stored in Amazon S3.

**Note**

A model package is specific to the region in which you create it. The S3 bucket where the model artifacts are stored must be in the same region where your created the model package.

2. Create a transform job in your account using the model to verify that your inference image works with SageMaker.
3. Create a validation profile.

**Note**

In your validation profile, provide only data that you want to expose publicly.

Validation can take up to a few hours. To see the status of the jobs in your account, in the SageMaker console, see the **Transform jobs** pages. If validation fails, you can access the scan and validation reports from the SageMaker console. After fixing issues, recreate the algorithm. When the status of the algorithm is **COMPLETED**, find it in the SageMaker console and start the listing process

**Note**

To publish your model package on Amazon Web Services Marketplace, at least one validation profile is required.

You can create a model package either by using the SageMaker console or by using the SageMaker API.

### Topics

- [Create a Model Package Resource \(Console\) \(p. 2401\)](#)
- [Create a Model Package Resource \(API\) \(p. 2402\)](#)

## Create a Model Package Resource (Console)

### To create a model package in the SageMaker console:

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Choose **Model packages**, then choose **Create model package**.
3. On the **Inference specifications** page, provide the following information:
  - a. For **Model package name**, type a name for your model package. The model package name must be unique in your account and in the Amazon region. The name must have 1 to 64 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).
  - b. Type a description for your model package. This description appears in the SageMaker console and in the Amazon Web Services Marketplace.
  - c. For **Inference specification options**, choose **Provide the location of the inference image and model artifacts** to create a model package by using an inference container and model artifacts. Choose **Provide the algorithm used for training and its model artifacts** to create a model package from an algorithm resource that you created or subscribe to from Amazon Web Services Marketplace.
  - d. If you chose **Provide the location of the inference image and model artifacts** for **Inference specification options**, provide the following information for **Container definition** and **Supported resources**:
    - i. For **Location of inference image**, type the path to the image that contains your inference code. The image must be stored as a Docker container in Amazon ECR.
    - ii. For **Location of model data artifacts**, type the location in S3 where your model artifacts are stored.
    - iii. For **Container DNS host name**, type the name of the DNS host to use for your container.
    - iv. For **Supported instance types for real-time inference**, choose the instance types that your model package supports for real-time inference from SageMaker hosted endpoints.
    - v. For **Supported instance types for batch transform jobs**, choose the instance types that your model package supports for batch transform jobs.
    - vi. For **Supported content types**, type the content types that your model package expects for inference requests.
    - vii. For **Supported response MIME types**, type the MIME types that your model package uses to provide inferences.
  - e. If you chose **Provide the algorithm used for training and its model artifacts** for **Inference specification options**, provide the following information:
    - i. For **Algorithm ARN**, type the Amazon Resource Name (ARN) of the algorithm resource to use to create the model package.
    - ii. For **Location of model data artifacts**, type the location in S3 where your model artifacts are stored.
  - f. Choose **Next**.
4. On the **Validation and scanning** page, provide the following information:
  - a. For **Publish this model package on Amazon Web Services Marketplace**, choose **Yes** to publish the model package on Amazon Web Services Marketplace.

- b. For **Validate this model package**, choose **Yes** if you want SageMaker to run batch transform jobs that you specify to test the inference code of your model package.

**Note**

To publish your model package on Amazon Web Services Marketplace, your model package must be validated.

- c. For **IAM role**, choose an IAM role that has the required permissions to run batch transform jobs in SageMaker, or choose **Create a new role** to allow SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see [SageMaker Roles \(p. 2447\)](#).
  - d. For **Validation profile**, specify the following:
    - A name for the validation profile.
    - A **Transform job definition**. This is a JSON block that describes a batch transform job. This is in the same format as the `TransformJobDefinition` input parameter of the `CreateAlgorithm` API.
5. Choose **Create model package**.

## Create a Model Package Resource (API)

To create a model package by using the SageMaker API, call the [CreateModelPackage](#) API.

# Use Algorithm and Model Package Resources

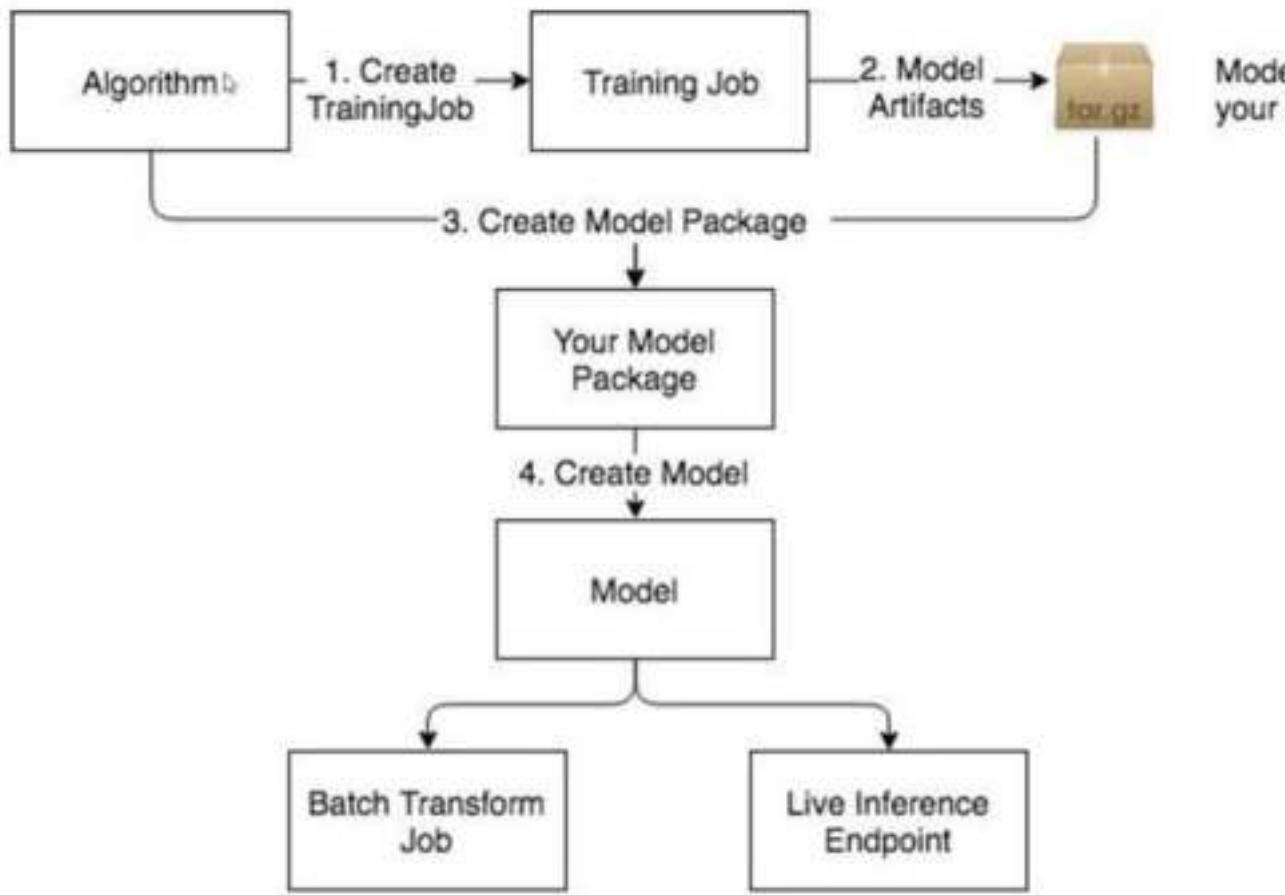
You can create algorithms and model packages as resources in your Amazon SageMaker account, and you can find and subscribe to algorithms and model packages on Amazon Web Services Marketplace.

Use algorithms to:

- Run training jobs. For information, see [Use an Algorithm to Run a Training Job \(p. 2403\)](#).
- Run hyperparameter tuning jobs. For information, see [Use an Algorithm to Run a Hyperparameter Tuning Job \(p. 2405\)](#).
- Create model packages. After you use an algorithm resource to run a training job or a hyperparameter tuning job, you can use the model artifacts that these jobs output along with the algorithm to create a model package. For information, see [Create a Model Package Resource \(p. 2400\)](#).

**Note**

If you subscribe to an algorithm on Amazon Web Services Marketplace, you must create a model package before you can use it to get inferences by creating hosted endpoint or running a batch transform job.



Use model packages to:

- Create models that you can use to get real-time inference or run batch transform jobs. For information, see [Use a Model Package to Create a Model \(p. 2408\)](#).
- Create hosted endpoints to get real-time inference. For information, see [Deploy the Model to SageMaker Hosting Services \(p. 63\)](#).
- Create batch transform jobs. For information, see [\(Optional\) Make Prediction with Batch Transform \(p. 64\)](#).

#### Topics

- [Use an Algorithm to Run a Training Job \(p. 2403\)](#)
- [Use an Algorithm to Run a Hyperparameter Tuning Job \(p. 2405\)](#)
- [Use a Model Package to Create a Model \(p. 2408\)](#)

## Use an Algorithm to Run a Training Job

You can create use an algorithm resource to create a training job by using the Amazon SageMaker console, the low-level Amazon SageMaker API, or the [Amazon SageMaker Python SDK](#).

#### Topics

- [Use an Algorithm to Run a Training Job \(Console\) \(p. 2404\)](#)
- [Use an Algorithm to Run a Training Job \(API\) \(p. 2405\)](#)

- [Use an Algorithm to Run a Training Job \(Amazon SageMaker Python SDK\) \(p. 2405\)](#)

## Use an Algorithm to Run a Training Job (Console)

### To use an algorithm to run a training job (console)

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Choose **Algorithms**.
3. Choose an algorithm that you created from the list on the **My algorithms** tab or choose an algorithm that you subscribed to on the **Amazon Web Services Marketplace subscriptions** tab.
4. Choose **Create training job**.

The algorithm you chose will automatically be selected.

5. On the **Create training job** page, provide the following information:
  - a. For **Job name**, type a name for the training job.
  - b. For **IAM role**, choose an IAM role that has the required permissions to run training jobs in SageMaker, or choose **Create a new role** to allow SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see [SageMaker Roles \(p. 2447\)](#).
  - c. For **Resource configuration**, provide the following information:
    - i. For **Instance type**, choose the instance type to use for training.
    - ii. For **Instance count**, type the number of ML instances to use for the training job.
    - iii. For **Additional volume per instance (GB)**, type the size of the ML storage volume that you want to provision. ML storage volumes store model artifacts and incremental states.
    - iv. For **Encryption key**, if you want Amazon SageMaker to use an Amazon Key Management Service key to encrypt data in the ML storage volume attached to the training instance, specify the key.
    - v. For **Stopping condition**, specify the maximum amount of time in seconds, minutes, hours, or days, that you want the training job to run.
  - d. For **VPC**, choose a Amazon VPC that you want to allow your training container to access. For more information, see [Give SageMaker Training Jobs Access to Resources in Your Amazon VPC \(p. 2513\)](#).
  - e. For **Hyperparameters**, specify the values of the hyperparameters to use for the training job.
  - f. For **Input data configuration**, specify the following values for each channel of input data to use for the training job. You can see what channels the algorithm you're using for training supports, and the content type, supported compression type, and supported input modes for each channel, under **Channel specification** section of the **Algorithm summary** page for the algorithm.
    - i. For **Channel name**, type the name of the input channel.
    - ii. For **Content type**, type the content type of the data that the algorithm expects for the channel.
    - iii. For **Compression type**, choose the data compression type to use, if any.
    - iv. For **Record wrapper**, choose RecordIO if the algorithm expects data in the RecordIO format.
    - v. For **S3 data type**, **S3 data distribution type**, and **S3 location**, specify the appropriate values. For information about what these values mean, see [S3DataSource](#).
    - vi. For **Input mode**, choose **File** to download the data from to the provisioned ML storage volume, and mount the directory to a Docker volume. Choose **Pipe** to stream data directly from Amazon S3 to the container.

- vii. To add another input channel, choose **Add channel**. If you are finished adding input channels, choose **Done**.
- g. For **Output** location, specify the following values:
  - i. For **S3 output path**, choose the S3 location where the training job stores output, such as model artifacts.

**Note**  
You use the model artifacts stored at this location to create a model or model package from this training job.
  - ii. For **Encryption key**, if you want SageMaker to use a Amazon KMS key to encrypt output data at rest in the S3 location.
- h. For **Tags**, specify one or more tags to manage the training job. Each tag consists of a key and an optional value. Tag keys must be unique per resource.
- i. Choose **Create training job** to run the training job.

## Use an Algorithm to Run a Training Job (API)

To use an algorithm to run a training job by using the SageMaker API, specify either the name or the Amazon Resource Name (ARN) as the `AlgorithmName` field of the `AlgorithmSpecification` object that you pass to `CreateTrainingJob`. For information about training models in SageMaker, see [Train a Model with Amazon SageMaker \(p. 8\)](#).

## Use an Algorithm to Run a Training Job (Amazon SageMaker Python SDK)

Use an algorithm that you created or subscribed to on Amazon Web Services Marketplace to create a training job, create an `AlgorithmEstimator` object and specify either the Amazon Resource Name (ARN) or the name of the algorithm as the value of the `algorithm_arn` argument. Then call the `fit` method of the estimator. For example:

```
from sagemaker import AlgorithmEstimator
data_path = os.path.join(DATA_DIR, 'marketplace', 'training')

algo = AlgorithmEstimator(
    algorithm_arn='arn:aws:sagemaker:us-east-2:012345678901:algorithm/my-algorithm',
    role='SageMakerRole',
    instance_count=1,
    instance_type='ml.c4.xlarge',
    sagemaker_session=sagemaker_session,
    base_job_name='test-marketplace')

train_input = algo.sagemaker_session.upload_data(
    path=data_path, key_prefix='integ-test-data/marketplace/train')

algo.fit({'training': train_input})
```

## Use an Algorithm to Run a Hyperparameter Tuning Job

A hyperparameter tuning job finds the best version of a model by running many training jobs on your dataset using the algorithm and ranges of hyperparameters that you specify. It then chooses the hyperparameter values that result in a model that performs the best, as measured by a metric that you choose. For more information, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

You can create use an algorithm resource to create a hyperparameter tuning job by using the Amazon SageMaker console, the low-level Amazon SageMaker API, or the [Amazon SageMaker Python SDK](#).

### Topics

- [Use an Algorithm to Run a Hyperparameter Tuning Job \(Console\) \(p. 2406\)](#)
- [Use an Algorithm to Run a Hyperparameter Tuning Job \(API\) \(p. 2407\)](#)
- [Use an Algorithm to Run a Hyperparameter Tuning Job \(Amazon SageMaker Python SDK\) \(p. 2408\)](#)

## Use an Algorithm to Run a Hyperparameter Tuning Job (Console)

### To use an algorithm to run a hyperparameter tuning job (console)

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
  2. Choose **Algorithms**.
  3. Choose an algorithm that you created from the list on the **My algorithms** tab or choose an algorithm that you subscribed to on the **Amazon Web Services Marketplace subscriptions** tab.
  4. Choose **Create hyperparameter tuning job**.
- The algorithm you chose will automatically be selected.
5. On the **Create hyperparameter tuning job** page, provide the following information:
    - a. For **Warm start**, choose **Enable warm start** to use the information from previous hyperparameter tuning jobs as a starting point for this hyperparameter tuning job. For more information, see [Run a Warm Start Hyperparameter Tuning Job \(p. 1765\)](#).
      - i. Choose **Identical data and algorithm** if your input data is the same as the input data for the parent jobs of this hyperparameter tuning job, or choose **Transfer learning** to use additional or different input data for this hyperparameter tuning job.
      - ii. For **Parent hyperparameter tuning job(s)**, choose up to 5 hyperparameter tuning jobs to use as parents to this hyperparameter tuning job.
    - b. For **Hyperparameter tuning job name**, type a name for the tuning job.
    - c. For **IAM role**, choose an IAM role that has the required permissions to run hyperparameter tuning jobs in SageMaker, or choose **Create a new role** to allow SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see [SageMaker Roles \(p. 2447\)](#).
    - d. For **VPC**, choose a Amazon VPC that you want to allow the training jobs that the tuning job launches to access. For more information, see [Give SageMaker Training Jobs Access to Resources in Your Amazon VPC \(p. 2513\)](#).
    - e. Choose **Next**.
    - f. For **Objective metric**, choose the metric that the hyperparameter tuning job uses to determine the best combination of hyperparameters, and choose whether to minimize or maximize this metric. For more information, see [View the Best Training Job \(p. 1763\)](#).
    - g. For **Hyperparameter configuration**, choose ranges for the tunable hyperparameters that you want the tuning job to search, and set static values for hyperparameters that you want to remain constant in all training jobs that the hyperparameter tuning job launches. For more information, see [Define Hyperparameter Ranges \(p. 1748\)](#).
    - h. Choose **Next**.
    - i. For **Input data configuration**, specify the following values for each channel of input data to use for the hyperparameter tuning job. You can see what channels the algorithm you're using for hyperparameter tuning supports, and the content type, supported compression type, and supported input modes for each channel, under **Channel specification** section of the **Algorithm summary** page for the algorithm.
      - i. For **Channel name**, type the name of the input channel.
      - ii. For **Content type**, type the content type of the data that the algorithm expects for the channel.
      - iii. For **Compression type**, choose the data compression type to use, if any.

- iv. For **Record wrapper**, choose RecordIO if the algorithm expects data in the RecordIO format.
  - v. For **S3 data type**, **S3 data distribution type**, and **S3 location**, specify the appropriate values. For information about what these values mean, see [S3DataSource](#).
  - vi. For **Input mode**, choose **File** to download the data from to the provisioned ML storage volume, and mount the directory to a Docker volume. Choose **Pipe** to stream data directly from Amazon S3 to the container.
  - vii. To add another input channel, choose **Add channel**. If you are finished adding input channels, choose **Done**.
- j. For **Output** location, specify the following values:
- i. For **S3 output path**, choose the S3 location where the training jobs that this hyperparameter tuning job launches store output, such as model artifacts.
- Note**  
You use the model artifacts stored at this location to create a model or model package from this hyperparameter tuning job.
- ii. For **Encryption key**, if you want SageMaker to use a Amazon KMS key to encrypt output data at rest in the S3 location.
- k. For **Resource configuration**, provide the following information:
- i. For **Instance type**, choose the instance type to use for each training job that the hyperparameter tuning job launches.
  - ii. For **Instance count**, type the number of ML instances to use for each training job that the hyperparameter tuning job launches.
  - iii. For **Additional volume per instance (GB)**, type the size of the ML storage volume that you want to provision each training job that the hyperparameter tuning job launches. ML storage volumes store model artifacts and incremental states.
  - iv. For **Encryption key**, if you want Amazon SageMaker to use an Amazon Key Management Service key to encrypt data in the ML storage volume attached to the training instances, specify the key.
- l. For **Resource limits**, provide the following information:
- i. For **Maximum training jobs**, specify the maximum number of training jobs that you want the hyperparameter tuning job to launch. A hyperparameter tuning job can launch a maximum of 500 training jobs.
  - ii. For **Maximum parallel training jobs**, specify the maximum number of concurrent training jobs that the hyperparameter tuning job can launch. A hyperparameter tuning job can launch a maximum of 10 concurrent training jobs.
  - iii. For **Stopping condition**, specify the maximum amount of time in seconds, minutes, hours, or days, that you want each training job that the hyperparameter tuning job launches to run.
  - m. For **Tags**, specify one or more tags to manage the hyperparameter tuning job. Each tag consists of a key and an optional value. Tag keys must be unique per resource.
  - n. Choose **Create jobs** to run the hyperparameter tuning job.

## Use an Algorithm to Run a Hyperparameter Tuning Job (API)

To use an algorithm to run a hyperparameter tuning job by using the SageMaker API, specify either the name or the Amazon Resource Name (ARN) of the algorithm as the `AlgorithmName` field of the `AlgorithmSpecification` object that you pass to [CreateHyperParameterTuningJob](#). For information about hyperparameter tuning in SageMaker, see [Perform Automatic Model Tuning with SageMaker \(p. 1745\)](#).

## Use an Algorithm to Run a Hyperparameter Tuning Job (Amazon SageMaker Python SDK)

Use an algorithm that you created or subscribed to on Amazon Web Services Marketplace to create a hyperparameter tuning job, create an `AlgorithmEstimator` object and specify either the Amazon Resource Name (ARN) or the name of the algorithm as the value of the `algorithm_arn` argument. Then initialize a `HyperparameterTuner` object with the `AlgorithmEstimator` you created as the value of the `estimator` argument. Finally, call the `fit` method of the `AlgorithmEstimator`. For example:

```
from sagemaker import AlgorithmEstimator
from sagemaker.tuner import HyperparameterTuner

data_path = os.path.join(DATA_DIR, 'marketplace', 'training')

algo = AlgorithmEstimator(
    algorithm_arn='arn:aws:sagemaker:us-east-2:764419575721:algorithm/scikit-decision-trees-1542410022',
    role='SageMakerRole',
    instance_count=1,
    instance_type='ml.c4.xlarge',
    sagemaker_session=sagemaker_session,
    base_job_name='test-marketplace')

train_input = algo.sagemaker_session.upload_data(
    path=data_path, key_prefix='integ-test-data/marketplace/train')

algo.set_hyperparameters(max_leaf_nodes=10)
tuner = HyperparameterTuner(estimator=algo, base_tuning_job_name='some-name',
                             objective_metric_name='validation:accuracy',
                             hyperparameter_ranges=hyperparameter_ranges,
                             max_jobs=2, max_parallel_jobs=2)

tuner.fit({'training': train_input}, include_cls_metadata=False)
tuner.wait()
```

## Use a Model Package to Create a Model

Use a model package to create a deployable model that you can use to get real-time inferences by creating a hosted endpoint or to run batch transform jobs. You can create a deployable model from a model package by using the Amazon SageMaker console, the low-level SageMaker API, or the [Amazon SageMaker Python SDK](#).

### Topics

- [Use a Model Package to Create a Model \(Console\) \(p. 2408\)](#)
- [Use a Model Package to Create a Model \(API\) \(p. 2409\)](#)
- [Use a Model Package to Create a Model \(Amazon SageMaker Python SDK\) \(p. 2409\)](#)

## Use a Model Package to Create a Model (Console)

### To create a deployable model from a model package (console)

1. Open the SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. Choose **Model packages**.
3. Choose a model package that you created from the list on the **My model packages** tab or choose a model package that you subscribed to on the **Amazon Web Services Marketplace subscriptions** tab.
4. Choose **Create model**.

5. For **Model name**, type a name for the model.
6. For **IAM role**, choose an IAM role that has the required permissions to call other services on your behalf, or choose **Create a new role** to allow SageMaker to create a role that has the `AmazonSageMakerFullAccess` managed policy attached. For information, see [SageMaker Roles \(p. 244\)](#).
7. For **VPC**, choose a Amazon VPC that you want to allow the model to access. For more information, see [Give SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC \(p. 2510\)](#).
8. Leave the default values for **Container input options** and **Choose model package**.
9. For environment variables, provide the names and values of environment variables you want to pass to the model container.
10. For **Tags**, specify one or more tags to manage the model. Each tag consists of a key and an optional value. Tag keys must be unique per resource.
11. Choose **Create model**.

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosting endpoints in SageMaker, see [Deploy Models for Inference](#).

## Use a Model Package to Create a Model (API)

To use a model package to create a deployable model by using the SageMaker API, specify the name or the Amazon Resource Name (ARN) of the model package as the `ModelPackageName` field of the `ContainerDefinition` object that you pass to the `CreateModel` API.

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosted endpoints in SageMaker, see [Deploy Models for Inference](#).

## Use a Model Package to Create a Model (Amazon SageMaker Python SDK)

To use a model package to create a deployable model by using the SageMaker Python SDK, initialize a `ModelPackage` object, and pass the Amazon Resource Name (ARN) of the model package as the `model_package_arn` argument. For example:

```
from sagemaker import ModelPackage
model = ModelPackage(role='SageMakerRole',
                      model_package_arn='training-job-scikit-decision-trees-1542660466-6f92',
                      sagemaker_session=sagemaker_session)
```

After you create a deployable model, you can use it to set up an endpoint for real-time inference or create a batch transform job to get inferences on entire datasets. For information about hosting endpoints in SageMaker, see [Deploy Models for Inference](#).

# Security in Amazon SageMaker

Cloud security at Amazon is the highest priority. As an Amazon customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between Amazon and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – Amazon is responsible for protecting the infrastructure that runs Amazon services in the Amazon Cloud. Amazon also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [Amazon compliance programs](#). To learn about the compliance programs that apply to Amazon SageMaker, see [Amazon Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the Amazon service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using SageMaker. The following topics show you how to configure SageMaker to meet your security and compliance objectives. You also learn how to use other Amazon services that help you to monitor and secure your SageMaker resources.

## Topics

- [Access Control \(p. 2410\)](#)
- [Data Protection in Amazon SageMaker \(p. 2412\)](#)
- [Identity and Access Management for Amazon SageMaker \(p. 2417\)](#)
- [Logging and Monitoring \(p. 2489\)](#)
- [Compliance Validation for Amazon SageMaker \(p. 2490\)](#)
- [Resilience in Amazon SageMaker \(p. 2490\)](#)
- [Infrastructure Security in Amazon SageMaker \(p. 2491\)](#)

## Access Control

Amazon SageMaker Studio notebooks and SageMaker notebook instances differ in their runtime environments. The following topics describe how to control root access to these notebooks.

## Topics

- [Access control and SageMaker Studio notebooks \(p. 2410\)](#)
- [Control root access to a SageMaker notebook instance \(p. 2412\)](#)

## Access control and SageMaker Studio notebooks

Amazon SageMaker Studio uses filesystem and container permissions for access control and isolation of Studio users and notebooks. This is one of the major differences between Studio notebooks and SageMaker notebook instances. This topic describes how permissions are set up to avoid security threats,

what SageMaker does by default, and how the customer can customize the permissions. For more information about Studio notebooks and their runtime environment, see [Use Amazon SageMaker Studio Notebooks \(p. 76\)](#).

### SageMaker app permissions

A *run-as user* is a POSIX user/group which is used to run the JupyterServer app and KernelGateway apps inside the container.

The run-as user for the JupyterServer app is sagemaker-user (1000) by default. This user has sudo permissions to enable the installation of dependencies such as yum packages.

The run-as user for the KernelGateway apps is root (0) by default. This user is able to install dependencies using pip/apt-get/conda.

Due to the previously mentioned user remapping, neither user is able to access resources or make changes to the host instance.

### User remapping

SageMaker performs user-remapping to map a user inside the container to a user on the host instance outside the container. The range of user IDs (0 - 65535) in the container are mapped to non-privileged user IDs above 65535 on the instance. For example, sagemaker-user (1000) inside the container might map to user (200001) on the instance, where the number in parentheses is the user ID. If the customer creates a new user/group inside the container, it won't be privileged on the host instance regardless of the user/group ID. The root user of the container is also mapped to a non-privileged user on the instance. For more information, see [Isolate containers with a user namespace](#).

### Custom image permissions

Customers can bring their own custom SageMaker images. These images can specify a different run-as user/group to launch the KernelGateway app. The customer can implement fine grained permission control inside the image, for example, to disable root access or perform other actions. The same user remapping applies here. For more information, see [Bring your own SageMaker image \(p. 95\)](#).

### Container isolation

Docker keeps a list of default capabilities that the container can use. SageMaker doesn't add additional capabilities. SageMaker adds specific route rules to block requests to Amazon EFS and the [instance metadata service](#) (IMDS) from the container. Customers can't change these route rules from the container. For more information, see [Runtime privilege and Linux capabilities](#).

### App metadata access

Metadata used by running apps are mounted to the container with read-only permission. Customers aren't able to modify this metadata from the container. For the available metadata, see [Get Notebook and App Metadata \(p. 84\)](#).

### User isolation on EFS

When you onboard to Studio, SageMaker creates an Amazon Elastic File System (EFS) volume for your domain that is shared by all Studio users in the domain. Each user gets their own private home directory on the EFS volume. This home directory is used to store the user's notebooks, Git repositories, and other data. To prevent other users in the domain from accessing the user's data, SageMaker creates a globally unique user ID for the user's profile and applies it as a POSIX user/group ID for the user's home directory.

### EBS access

An Amazon Elastic Block Store (Amazon EBS) volume is attached to the host instance and shared across all images. It's used for the root volume of the notebooks and stores temporary data that's generated

inside the container. The storage isn't persisted when the instance running the notebooks is deleted. The root user inside the container can't access the EBS volume.

#### IMDS access

Due to security concerns, access to the Amazon Elastic Compute Cloud (Amazon EC2) Instance Metadata Service (IMDS) is unavailable in SageMaker Studio. For more information on IMDS, see [Instance metadata and user data](#).

## Control root access to a SageMaker notebook instance

By default, when you create a notebook instance, users that log into that notebook instance have root access. Data science is an iterative process that might require the data scientist to test and use different software tools and packages, so many notebook instance users need to have root access to be able to install these tools and packages. Because users with root access have administrator privileges, users can access and edit all files on a notebook instance with root access enabled.

If you don't want users to have root access to a notebook instance, when you call [CreateNotebookInstance](#) or [UpdateNotebookInstance](#) operations, set the `RootAccess` field to `Disabled`. You can also disable root access for users when you create or update a notebook instance in the Amazon SageMaker console. For information, see [Step 1: Create an Amazon SageMaker Notebook Instance \(p. 53\)](#).

#### Note

Lifecycle configurations need root access to be able to set up a notebook instance. Because of this, lifecycle configurations associated with a notebook instance always run with root access even if you disable root access for users.

#### Note

For security reasons, Rootless Docker is installed on root-disabled notebook instances instead of regular Docker. For more information, see [Run the Docker daemon as a non-root user \(Rootless mode\)](#)

## Data Protection in Amazon SageMaker

The Amazon [shared responsibility model](#) applies to data protection in Amazon SageMaker. As described in this model, Amazon is responsible for protecting the global infrastructure that runs all of the Amazon Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the Amazon services that you use. For more information about data privacy, see the [Data Privacy FAQ](#).

For data protection purposes, we recommend that you protect Amazon account credentials and set up individual user accounts with Amazon Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with Amazon resources. We recommend TLS 1.2 or later.
- Set up API and user activity logging with Amazon CloudTrail.
- Use Amazon encryption solutions, along with all default security controls within Amazon services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

- If you require FIPS 140-2 validated cryptographic modules when accessing Amazon through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with Amazon SageMaker or other Amazon services using the console, API, Amazon CLI, or Amazon SDKs. Any data that you enter into Amazon SageMaker or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

#### Topics

- [Protect Data at Rest Using Encryption \(p. 2413\)](#)
- [Protecting Data in Transit with Encryption \(p. 2414\)](#)
- [Key Management \(p. 2416\)](#)
- [Internetwork Traffic Privacy \(p. 2417\)](#)

## Protect Data at Rest Using Encryption

To protect your Amazon SageMaker Studio notebooks and SageMaker notebook instances, along with your model-building data and model artifacts, SageMaker uses the Amazon Key Management Service (Amazon KMS) to encrypt the notebooks and data. SageMaker uses Amazon managed customer master keys (CMKs) by default. For more control, you can specify your own customer managed CMKs. For more information on Amazon KMS, see [What is Amazon Key Management Service?](#).

#### Topics

- [Studio notebooks \(p. 2413\)](#)
- [Notebook instances and SageMaker jobs \(p. 2414\)](#)

## Studio notebooks

In Amazon SageMaker Studio, your SageMaker Studio notebooks and data can be stored in the following locations:

- An S3 bucket – When you onboard to Studio and enable shareable notebook resources, SageMaker shares notebook snapshots and metadata in an Amazon Simple Storage Service (Amazon S3) bucket.
- An EFS volume – When you onboard to Studio, SageMaker attaches an Amazon Elastic File System (Amazon EFS) volume to your domain for storing your Studio notebooks and data files. The EFS volume persists after the domain is deleted.
- An EBS volume – When you open a notebook in Studio, an Amazon Elastic Block Store (Amazon EBS) is attached to the instance that the notebook runs on. The EBS volume persists for the duration of the instance.

SageMaker uses the Amazon Key Management Service (Amazon KMS) to encrypt the S3 bucket and both volumes with an Amazon managed customer master key (CMK) by default. For more control, you can specify your own customer managed CMK when you onboard to Studio or through the SageMaker API. For more information, see [Onboard to Amazon SageMaker Studio \(p. 35\)](#) and [CreateDomain](#).

In the `CreateDomain` API, you use the `S3KmsKeyId` parameter to specify the custom CMK for shareable notebooks. You use the `KmsKeyId` parameter to specify the custom CMK for the EFS and EBS volumes. The same CMK is used for both volumes. The custom CMK for shareable notebooks can be the same CMK as used for the volumes or a different CMK.

## Notebook instances and SageMaker jobs

To encrypt the machine learning (ML) storage volume that is attached to notebooks, processing jobs, training jobs, hyperparameter tuning jobs, batch transform jobs, and endpoints, you can pass a Amazon KMS key to SageMaker. If you don't specify a KMS key, SageMaker encrypts storage volumes with a transient key and discards it immediately after encrypting the storage volume. For notebook instances, if you don't specify a KMS key, SageMaker encrypts both OS volumes and ML data volumes with a system-managed KMS key.

You can use an Amazon managed KMS key to encrypt all instance OS volumes. You can encrypt all ML data volumes for all SageMaker instances with a KMS key that you specify. ML storage volumes are mounted as follows:

- Notebooks - /home/ec2-user/SageMaker
- Processing - /opt/ml/processing and /tmp/
- Training - /opt/ml/ and /tmp/
- Batch - /opt/ml/ and /tmp/
- Endpoints - /opt/ml/ and /tmp/

Processing, batch transform, and training job containers and their storage are ephemeral in nature. When the job completes, output is uploaded to Amazon S3 using Amazon KMS encryption (with an optional KMS key you specify) and the instance is torn down.

### Important

Sensitive data that needs to be encrypted with a KMS key for compliance reasons should be stored in the ML storage volume or in Amazon S3, both of which can be encrypted using a KMS key you specify.

When you open a notebook instance, SageMaker saves it and any files associated with it in the SageMaker folder in the ML storage volume by default. When you stop a notebook instance, SageMaker creates a snapshot of the ML storage volume. Any customizations to the operating system of the stopped instance, such as installed custom libraries or operating system level settings, are lost. Consider using a lifecycle configuration to automate customizations of the default notebook instance. When you terminate an instance, the snapshot and the ML storage volume are deleted. Any data that you need to persist beyond the lifespan of the notebook instance should be transferred to an Amazon S3 bucket.

### Note

Certain Nitro-based SageMaker instances include local storage, depending on the instance type. Local storage volumes are encrypted using a hardware module on the instance. You can't use a KMS key on an instance type with local storage. For a list of instance types that support local instance storage, see [Instance Store Volumes](#). For more information about storage volumes on Nitro-based instances, see [Amazon EBS and NVMe on Linux Instances](#).

For more information about local instance storage encryption, see [SSD Instance Store Volumes](#).

## Protecting Data in Transit with Encryption

All inter-network data in transit supports TLS 1.2 encryption.

Amazon SageMaker ensures that machine learning (ML) model artifacts and other system artifacts are encrypted in transit and at rest. Requests to the SageMaker API and console are made over a secure (SSL) connection. You pass Amazon Identity and Access Management roles to SageMaker to provide permissions to access resources on your behalf for training and deployment. You can use encrypted Amazon S3 buckets for model artifacts and data, as well as pass a Amazon KMS key to SageMaker instances to encrypt the attached ML storage volumes.

Some intra-network data in-transit (inside the service platform) is unencrypted. This includes:

- Command and control communications between the service control plane and training job instances (not customer data).
- Communications between nodes in distributed processing jobs (intra-network).
- Communications between nodes in distributed training jobs (intra-network).

There are no inter-node communications for batch processing.

You can choose to encrypt communication between nodes in a training cluster. For information about how to do this, see [Protect Communications Between ML Compute Instances in a Distributed Training Job \(p. 2415\)](#). Enabling inter-container traffic encryption can increase training time, especially if you are using distributed deep learning algorithms. For affected algorithms, adding this additional level of security also increases cost. The training time for most SageMaker built-in algorithms, such as XGBoost, DeepAR, and linear learner, typically aren't affected.

FIPS validated endpoints are available for the SageMaker API and request router for hosted models (runtime). For information about FIPS compliant endpoints, see [Federal Information Processing Standard \(FIPS\) 140-2](#).

## Protect Communications Between ML Compute Instances in a Distributed Training Job

By default, Amazon SageMaker runs training jobs in an Amazon Virtual Private Cloud (Amazon VPC) to help keep your data secure. You can add another level of security to protect your training containers and data by configuring a *private* VPC. Distributed ML frameworks and algorithms usually transmit information that is directly related to the model such as weights, not the training dataset. When performing distributed training, you can further protect data that is transmitted between instances. This can help you to comply with regulatory requirements. To do this, use inter-container traffic encryption.

Enabling inter-container traffic encryption can increase training time, especially if you are using distributed deep learning algorithms. Enabling inter-container traffic encryption doesn't affect training jobs with a single compute instance. However, for training jobs with several compute instances, the effect on training time depends on the amount of communication between compute instances. For affected algorithms, adding this additional level of security also increases cost. The training time for most SageMaker built-in algorithms, such as XGBoost, DeepAR, and linear learner, typically aren't affected.

You can enable inter-container traffic encryption for training jobs or hyperparameter tuning jobs. You can use SageMaker APIs or console to enable inter-container traffic encryption.

For information about running training jobs in a private VPC, see [Give SageMaker Training Jobs Access to Resources in Your Amazon VPC \(p. 2513\)](#).

### Enable Inter-Container Traffic Encryption (API)

Before enabling inter-container traffic encryption on training or hyperparameter tuning jobs with APIs, you need to add inbound and outbound rules to your private VPC's security group.

#### To enable inter-container traffic encryption (API)

1. Add the following inbound and outbound rules in the security group for your private VPC:

Protocol	Port Range	Source
UDP	500	<i>Self Security Group ID</i>
ESP 50	N/A	<i>Self Security Group ID</i>

2. When you send a request to the [CreateTrainingJob](#) or [CreateHyperParameterTuningJob](#) API, specify `True` for the `EnableInterContainerTrafficEncryption` parameter.

**Note**

For the `ESP 50` protocol, the Amazon Security Group Console might display the port range as "All". However, Amazon EC2 ignores the specified port range because it is not applicable for the `ESP 50` IP protocol.

## Enable Inter-Container Traffic Encryption (Console)

### Enable Inter-container Traffic Encryption in a Training Job

#### To enable inter-container traffic encryption in a training job

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the navigation pane, choose **Training**, then choose **Training jobs**.
3. Choose **Create training job**.
4. Under **Network**, choose a **VPC**. You can use the default VPC or one that you have created.
5. Choose **Enable inter-container traffic encryption**.

After you enable inter-container traffic encryption, finish creating the training job. For more information, see [Step 4: Train a Model \(p. 59\)](#).

### Enable Inter-container Traffic Encryption in a Hyperparameter Tuning Job

#### To enable inter-container traffic encryption in a hyperparameter tuning job

1. Open the Amazon SageMaker console at <https://console.amazonaws.cn/sagemaker/>.
2. In the navigation pane, choose **Training**, then choose **Hyperparameter tuning jobs**.
3. Choose **Create hyperparameter tuning job**.
4. Under **Network**, choose a **VPC**. You can use the default VPC or one that you created.
5. Choose **Enable inter-container traffic encryption**.

After enabling inter-container traffic encryption, finish creating the hyperparameter tuning job. For more information, see [Configure and Launch a Hyperparameter Tuning Job \(p. 1757\)](#).

## Key Management

Customers can specify Amazon KMS keys, including bring your own keys (BYOK), to use for envelope encryption with Amazon S3 input/output buckets and machine learning (ML) Amazon EBS volumes. ML volumes for notebook instances and for processing, training, and hosted model Docker containers can be optionally encrypted by using Amazon KMS customer-owned keys. All instance OS volumes are encrypted with an Amazon-managed Amazon KMS key.

**Note**

Certain Nitro-based instances include local storage, dependent on the instance type. Local storage volumes are encrypted using a hardware module on the instance. You can't request a `VolumeKmsKeyId` when using an instance type with local storage.

For a list of instance types that support local instance storage, see [Instance Store Volumes](#).

For more information about local instance storage encryption, see [SSD Instance Store Volumes](#).

For more information about storage volumes on nitro-based instances, see [Amazon EBS and NVMe on Linux Instances](#).

For information about Amazon KMS keys see [What is Amazon Key Management Service?](#) in the *Amazon Key Management Service Developer Guide*.

## Internetwork Traffic Privacy

This topic describes how Amazon SageMaker secures connections from the service to other locations.

Internetwork communications support TLS 1.2 encryption between all components and clients.

Instances can be connected to Customer VPC, providing access to S3 VPC endpoints or customer repositories. Internet egress can be managed through this interface by the customer if service platform internet egress is disabled for notebooks. For training and hosting, egress through the service platform is not available when connected to the customer's VPC.

By default, API calls made to published endpoints traverse the public network to the request router. SageMaker supports Amazon Virtual Private Cloud interface endpoints powered by Amazon PrivateLink for private connectivity between the customer's VPC and the request router to access hosted model endpoints. For information about Amazon VPC, see [Connect to SageMaker Through a VPC Interface Endpoint \(p. 2495\)](#)

## Identity and Access Management for Amazon SageMaker

Amazon Identity and Access Management (IAM) is an Amazon service that helps an administrator securely control access to Amazon resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use SageMaker resources. IAM is an Amazon service that you can use with no additional charge.

### Topics

- [Audience \(p. 2417\)](#)
- [Authenticating with Identities \(p. 2418\)](#)
- [Managing Access Using Policies \(p. 2420\)](#)
- [How Amazon SageMaker Works with IAM \(p. 2421\)](#)
- [Amazon SageMaker Identity-Based Policy Examples \(p. 2424\)](#)
- [SageMaker Roles \(p. 2447\)](#)
- [Amazon Managed Policies for Amazon SageMaker \(p. 2464\)](#)
- [Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference \(p. 2474\)](#)
- [Troubleshooting Amazon SageMaker Identity and Access \(p. 2487\)](#)

## Audience

How you use Amazon Identity and Access Management (IAM) differs, depending on the work that you do in SageMaker.

**Service user** – If you use the SageMaker service to do your job, then your administrator provides you with the credentials and permissions that you need. As you use more SageMaker features to do your work, you might need additional permissions. Understanding how access is managed can help you request the right permissions from your administrator. If you cannot access a feature in SageMaker, see [Troubleshooting Amazon SageMaker Identity and Access \(p. 2487\)](#).

**Service administrator** – If you're in charge of SageMaker resources at your company, you probably have full access to SageMaker. It's your job to determine which SageMaker features and resources your employees should access. You must then submit requests to your IAM administrator to change the permissions of your service users. Review the information on this page to understand the basic concepts of IAM. To learn more about how your company can use IAM with SageMaker, see [How Amazon SageMaker Works with IAM \(p. 2421\)](#).

**IAM administrator** – If you're an IAM administrator, you might want to learn details about how you can write policies to manage access to SageMaker. To view example SageMaker identity-based policies that you can use in IAM, see [Amazon SageMaker Identity-Based Policy Examples \(p. 2424\)](#).

## Authenticating with Identities

Authentication is how you sign in to Amazon using your identity credentials. For more information about signing in using the Amazon Web Services Management Console, see [Signing in to the Amazon Web Services Management Console as an IAM user or root user](#) in the *IAM User Guide*.

You must be *authenticated* (signed in to Amazon) as the Amazon account root user, an IAM user, or by assuming an IAM role. You can also use your company's single sign-on authentication or even sign in using Google or Facebook. In these cases, your administrator previously set up identity federation using IAM roles. When you access Amazon using credentials from another company, you are assuming a role indirectly.

To sign in directly to the [Amazon Web Services Management Console](#), use your password with your root user email address or your IAM user name. You can access Amazon programmatically using your root user or IAM users access keys. Amazon provides SDK and command line tools to cryptographically sign your request using your credentials. If you don't use Amazon tools, you must sign the request yourself. Do this using *Signature Version 4*, a protocol for authenticating inbound API requests. For more information about authenticating requests, see [Signature Version 4 signing process](#) in the *Amazon General Reference*.

Regardless of the authentication method that you use, you might also be required to provide additional security information. For example, Amazon recommends that you use multi-factor authentication (MFA) to increase the security of your account. To learn more, see [Using multi-factor authentication \(MFA\) in Amazon](#) in the *IAM User Guide*.

### Amazon account root user

When you first create an Amazon account, you begin with a single sign-in identity that has complete access to all Amazon services and resources in the account. This identity is called the Amazon account *root user* and is accessed by signing in with the email address and password that you used to create the account. We strongly recommend that you do not use the root user for your everyday tasks, even the administrative ones. Instead, adhere to the [best practice of using the root user only to create your first IAM user](#). Then securely lock away the root user credentials and use them to perform only a few account and service management tasks.

### IAM Users and Groups

An *IAM user* is an identity within your Amazon account that has specific permissions for a single person or application. An IAM user can have long-term credentials such as a user name and password or a set of access keys. To learn how to generate access keys, see [Managing access keys for IAM users](#) in the *IAM User Guide*. When you generate access keys for an IAM user, make sure you view and securely save the key pair. You cannot recover the secret access key in the future. Instead, you must generate a new access key pair.

An *IAM group* is an identity that specifies a collection of IAM users. You can't sign in as a group. You can use groups to specify permissions for multiple users at a time. Groups make permissions easier to

manage for large sets of users. For example, you could have a group named *IAMAdmins* and give that group permissions to administer IAM resources.

Users are different from roles. A user is uniquely associated with one person or application, but a role is intended to be assumable by anyone who needs it. Users have permanent long-term credentials, but roles provide temporary credentials. To learn more, see [When to create an IAM user \(instead of a role\)](#) in the *IAM User Guide*.

## IAM Roles

An *IAM role* is an identity within your Amazon account that has specific permissions. It is similar to an IAM user, but is not associated with a specific person. You can temporarily assume an IAM role in the Amazon Web Services Management Console by [switching roles](#). You can assume a role by calling an Amazon CLI or Amazon API operation or by using a custom URL. For more information about methods for using roles, see [Using IAM roles](#) in the *IAM User Guide*.

IAM roles with temporary credentials are useful in the following situations:

- **Temporary IAM user permissions** – An IAM user can assume an IAM role to temporarily take on different permissions for a specific task.
- **Federated user access** – Instead of creating an IAM user, you can use existing identities from Amazon Directory Service, your enterprise user directory, or a web identity provider. These are known as *federated users*. Amazon assigns a role to a federated user when access is requested through an [identity provider](#). For more information about federated users, see [Federated users and roles](#) in the *IAM User Guide*.
- **Cross-account access** – You can use an IAM role to allow someone (a trusted principal) in a different account to access resources in your account. Roles are the primary way to grant cross-account access. However, with some Amazon services, you can attach a policy directly to a resource (instead of using a role as a proxy). To learn the difference between roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.
- **Cross-service access** – Some Amazon services use features in other Amazon services. For example, when you make a call in a service, it's common for that service to run applications in Amazon EC2 or store objects in Amazon S3. A service might do this using the calling principal's permissions, using a service role, or using a service-linked role.
  - **Principal permissions** – When you use an IAM user or role to perform actions in Amazon, you are considered a principal. Policies grant permissions to a principal. When you use some services, you might perform an action that then triggers another action in a different service. In this case, you must have permissions to perform both actions. To see whether an action requires additional dependent actions in a policy, see [Actions, Resources, and Condition Keys for Amazon SageMaker](#) in the *Service Authorization Reference*.
  - **Service role** – A service role is an *IAM role* that a service assumes to perform actions on your behalf. Service roles provide access only within your account and cannot be used to grant access to services in other accounts. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see [Creating a role to delegate permissions to an Amazon service](#) in the *IAM User Guide*.
  - **Service-linked role** – A service-linked role is a type of service role that is linked to an Amazon service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your IAM account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.
- **Applications running on Amazon EC2** – You can use an IAM role to manage temporary credentials for applications that are running on an EC2 instance and making Amazon CLI or Amazon API requests. This is preferable to storing access keys within the EC2 instance. To assign an Amazon role to an EC2 instance and make it available to all of its applications, you create an instance profile that is attached to the instance. An instance profile contains the role and enables programs that are running on the EC2 instance to get temporary credentials. For more information, see [Using an IAM role to grant permissions to applications running on Amazon EC2 instances](#) in the *IAM User Guide*.

To learn whether to use IAM roles or IAM users, see [When to create an IAM role \(instead of a user\)](#) in the *IAM User Guide*.

## Managing Access Using Policies

You control access in Amazon by creating policies and attaching them to IAM identities or Amazon resources. A policy is an object in Amazon that, when associated with an identity or resource, defines their permissions. You can sign in as the root user or an IAM user, or you can assume an IAM role. When you then make a request, Amazon evaluates the related identity-based or resource-based policies. Permissions in the policies determine whether the request is allowed or denied. Most policies are stored in Amazon as JSON documents. For more information about the structure and contents of JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

Every IAM entity (user or role) starts with no permissions. In other words, by default, users can do nothing, not even change their own password. To give a user permission to do something, an administrator must attach a permissions policy to a user. Or the administrator can add the user to a group that has the intended permissions. When an administrator gives permissions to a group, all users in that group are granted those permissions.

IAM policies define permissions for an action regardless of the method that you use to perform the operation. For example, suppose that you have a policy that allows the `iam:GetRole` action. A user with that policy can get role information from the Amazon Web Services Management Console, the Amazon CLI, or the Amazon API.

### Identity-Based Policies

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Creating IAM policies](#) in the *IAM User Guide*.

Identity-based policies can be further categorized as *inline policies* or *managed policies*. Inline policies are embedded directly into a single user, group, or role. Managed policies are standalone policies that you can attach to multiple users, groups, and roles in your Amazon account. Managed policies include Amazon managed policies and customer managed policies. To learn how to choose between a managed policy or an inline policy, see [Choosing between managed policies and inline policies](#) in the *IAM User Guide*.

### Resource-Based Policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are *IAM role trust policies* and *Amazon S3 bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or Amazon services.

Resource-based policies are inline policies that are located in that service. You can't use Amazon managed policies from IAM in a resource-based policy.

### Access Control Lists (ACLs)

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Amazon S3, Amazon WAF, and Amazon VPC are examples of services that support ACLs. To learn more about ACLs, see [Access control list \(ACL\) overview](#) in the *Amazon Simple Storage Service Developer Guide*.

## Other Policy Types

Amazon supports additional, less-common policy types. These policy types can set the maximum permissions granted to you by the more common policy types.

- **Permissions boundaries** – A permissions boundary is an advanced feature in which you set the maximum permissions that an identity-based policy can grant to an IAM entity (IAM user or role). You can set a permissions boundary for an entity. The resulting permissions are the intersection of entity's identity-based policies and its permissions boundaries. Resource-based policies that specify the user or role in the `Principal` field are not limited by the permissions boundary. An explicit deny in any of these policies overrides the allow. For more information about permissions boundaries, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – SCPs are JSON policies that specify the maximum permissions for an organization or organizational unit (OU) in Amazon Organizations. Amazon Organizations is a service for grouping and centrally managing multiple Amazon accounts that your business owns. If you enable all features in an organization, then you can apply service control policies (SCPs) to any or all of your accounts. The SCP limits permissions for entities in member accounts, including each Amazon account root user. For more information about Organizations and SCPs, see [How SCPs work](#) in the *Amazon Organizations User Guide*.
- **Session policies** – Session policies are advanced policies that you pass as a parameter when you programmatically create a temporary session for a role or federated user. The resulting session's permissions are the intersection of the user or role's identity-based policies and the session policies. Permissions can also come from a resource-based policy. An explicit deny in any of these policies overrides the allow. For more information, see [Session policies](#) in the *IAM User Guide*.

## Multiple Policy Types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how Amazon determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

## How Amazon SageMaker Works with IAM

Before you use IAM to manage access to SageMaker, you should understand what IAM features are available to use with SageMaker. To get a high-level view of how SageMaker and other Amazon services work with IAM, see [Amazon Services That Work with IAM](#) in the *IAM User Guide*.

### Topics

- [SageMaker Identity-Based Policies](#) (p. 2421)

## SageMaker Identity-Based Policies

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. SageMaker supports specific actions, resources, and condition keys. To learn about all of the elements that you use in a JSON policy, see [IAM JSON Policy Elements Reference](#) in the *IAM User Guide*.

### Actions

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The **Action** element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Policy actions usually have the same name as the associated Amazon API operation. There are some exceptions, such as *permission-only actions* that don't have a matching API operation. There are also some operations that require multiple actions in a policy. These additional actions are called *dependent actions*.

Include actions in a policy to grant permissions to perform the associated operation.

Policy actions in SageMaker use the following prefix before the action: `sagemaker::`. For example, to grant someone permission to run a SageMaker training job with the SageMaker `CreateTrainingJob` API operation, you include the `sagemaker:CreateTrainingJob` action in their policy. Policy statements must include either an **Action** or **NotAction** element. SageMaker defines its own set of actions that describe tasks that you can perform with this service.

To specify multiple actions in a single statement, separate them with commas as follows:

```
"Action": [  
    "sagemaker:action1",  
    "sagemaker:action2"  
]
```

You can specify multiple actions using wildcards (\*). For example, to specify all actions that begin with the word `Describe`, include the following action:

```
"Action": "sagemaker:Describe*"
```

To see a list of SageMaker actions, see [Actions, resources, and condition keys for Amazon SageMaker](#) in the *Service Authorization Reference*.

## Resources

SageMaker does not support specifying resource ARNs in a policy.

## Condition Keys

Administrators can use Amazon JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The **Condition** element (or **Condition block**) lets you specify conditions in which a statement is in effect. The **Condition** element is optional. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request.

If you specify multiple **Condition** elements in a statement, or multiple keys in a single **Condition** element, Amazon evaluates them using a logical **AND** operation. If you specify multiple values for a single condition key, Amazon evaluates the condition using a logical **OR** operation. All of the conditions must be met before the statement's permissions are granted.

You can also use placeholder variables when you specify conditions. For example, you can grant an IAM user permission to access a resource only if it is tagged with their IAM user name. For more information, see [IAM policy elements: variables and tags](#) in the *IAM User Guide*.

Amazon supports global condition keys and service-specific condition keys. To see all Amazon global condition keys, see [Amazon global condition context keys](#) in the *IAM User Guide*.

SageMaker defines its own set of condition keys and also supports using some global condition keys. To see all Amazon global condition keys, see [Amazon Global Condition Context Keys](#) in the *IAM User Guide*.

SageMaker supports a number of service-specific condition keys that you can use for fine-grained access control for the following operations:

- [CreateProcessingJob](#)
- [CreateTrainingJob](#)
- [CreateModel](#)
- [CreateEndpointConfig](#)
- [CreateTransformJob](#)
- [CreateHyperParameterTuningJob](#)
- [CreateLabelingJob](#)
- [CreateNotebookInstance](#)
- [UpdateNotebookInstance](#)

To see a list of SageMaker condition keys, see [Condition keys for Amazon SageMaker](#) in the *IAM User Guide*. To learn with which actions and resources you can use a condition key, see [Actions defined by Amazon SageMaker](#).

For examples of using SageMaker condition keys, see the following: [Control Creation of SageMaker Resources with Condition Keys \(p. 2434\)](#).

## Examples

To view examples of SageMaker identity-based policies, see [Amazon SageMaker Identity-Based Policy Examples \(p. 2424\)](#).

## SageMaker Resource-Based Policies

SageMaker does not support resource-based policies.

## Authorization Based on SageMaker Tags

You can attach tags to SageMaker resources or pass tags in a request to SageMaker. To control access based on tags, you provide tag information in the `condition element` of a policy using the `sagemaker:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys. For more information about tagging SageMaker resources, see [Control Access to SageMaker Resources by Using Tags \(p. 2444\)](#).

To view an example identity-based policy for limiting access to a resource based on the tags on that resource, see [Control Access to SageMaker Resources by Using Tags \(p. 2444\)](#).

## SageMaker IAM Roles

An [IAM role](#) is an entity within your Amazon account that has specific permissions.

### Using Temporary Credentials with SageMaker

You can use temporary credentials to sign in with federation, assume an IAM role, or to assume a cross-account role. You obtain temporary security credentials by calling Amazon STS API operations such as `AssumeRole` or `GetFederationToken`.

SageMaker supports using temporary credentials.

### Service-Linked Roles

SageMaker doesn't support service-linked roles.

## Service Roles

This feature allows a service to assume a [service role](#) on your behalf. This role allows the service to access resources in other services to complete an action on your behalf. Service roles appear in your IAM account and are owned by the account. This means that an IAM administrator can change the permissions for this role. However, doing so might break the functionality of the service.

SageMaker supports service roles.

### Choosing an IAM Role in SageMaker

When you create a notebook instance, processing job, training job, hosted endpoint, or batch transform job resource in SageMaker, you must choose a role to allow SageMaker to access SageMaker on your behalf. If you have previously created a service role or service-linked role, then SageMaker provides you with a list of roles to choose from. It's important to choose a role that allows access to the Amazon operations and resources you need. For more information, see [SageMaker Roles \(p. 2447\)](#).

## Amazon SageMaker Identity-Based Policy Examples

By default, IAM users and roles don't have permission to create or modify SageMaker resources. They also can't perform tasks using the Amazon Web Services Management Console, Amazon CLI, or Amazon API. An IAM administrator must create IAM policies that grant users and roles permission to perform specific API operations on the specified resources they need. The administrator must then attach those policies to the IAM users or groups that require those permissions. To learn how to attach policies to an IAM user or group, see [Adding and Removing IAM Identity Permissions](#) in the *IAM User Guide*.

To learn how to create an IAM identity-based policy using these example JSON policy documents, see [Creating Policies on the JSON Tab](#) in the *IAM User Guide*.

### Topics

- [Policy Best Practices \(p. 2424\)](#)
- [Using the SageMaker Console \(p. 2425\)](#)
- [Allow Users to View Their Own Permissions \(p. 2433\)](#)
- [Control Creation of SageMaker Resources with Condition Keys \(p. 2434\)](#)
- [Control Access to the SageMaker API by Using Identity-based Policies \(p. 2441\)](#)
- [Limit Access to SageMaker API and Runtime Calls by IP Address \(p. 2442\)](#)
- [Limit Access to a Notebook Instance by IP Address \(p. 2443\)](#)
- [Control Access to SageMaker Resources by Using Tags \(p. 2444\)](#)
- [Require the Presence or Absence of Tags for API Calls \(p. 2446\)](#)
- [Use Tags with Hyperparameter Tuning Jobs \(p. 2446\)](#)

## Policy Best Practices

Identity-based policies are very powerful. They determine whether someone can create, access, or delete SageMaker resources in your account. These actions can incur costs for your Amazon account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started using Amazon managed policies** – To start using SageMaker quickly, use Amazon managed policies to give your employees the permissions they need. These policies are already available in your account and are maintained and updated by Amazon. For more information, see [Get started using permissions with Amazon managed policies](#) in the *IAM User Guide*.
- **Grant least privilege** – When you create custom policies, grant only the permissions required to perform a task. Start with a minimum set of permissions and grant additional permissions as

necessary. Doing so is more secure than starting with permissions that are too lenient and then trying to tighten them later. For more information, see [Grant least privilege](#) in the *IAM User Guide*.

- **Enable MFA for sensitive operations** – For extra security, require IAM users to use multi-factor authentication (MFA) to access sensitive resources or API operations. For more information, see [Using multi-factor authentication \(MFA\) in Amazon](#) in the *IAM User Guide*.
- **Use policy conditions for extra security** – To the extent that it's practical, define the conditions under which your identity-based policies allow access to a resource. For example, you can write conditions to specify a range of allowable IP addresses that a request must come from. You can also write conditions to allow requests only within a specified date or time range, or to require the use of SSL or MFA. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.

## Using the SageMaker Console

To access the Amazon SageMaker console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the SageMaker resources in your Amazon account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (IAM users or roles) with that policy.

To ensure that those entities can still use the SageMaker console, also attach the following Amazon managed policy to the entities. For more information, see [Adding Permissions to a User](#) in the *IAM User Guide*:

You don't need to allow minimum console permissions for users that are making calls only to the Amazon CLI or the Amazon API. Instead, allow access to only the actions that match the API operation that you're trying to perform.

### Topics

- [Permissions Required to Use the Amazon SageMaker Console \(p. 2425\)](#)
- [Permissions Required to Use the Amazon SageMaker Ground Truth Console \(p. 2427\)](#)
- [Permissions Required to Use the Amazon Augmented AI \(Preview\) Console \(p. 2428\)](#)

## Permissions Required to Use the Amazon SageMaker Console

The permissions reference table lists the Amazon SageMaker API operations and shows the required permissions for each operation. For more information about Amazon SageMaker API operations, see [Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference \(p. 2474\)](#).

To use the Amazon SageMaker console, you need to grant permissions for additional actions. Specifically, the console needs permissions that allow the `ec2` actions to display subnets, VPCs, and security groups. Optionally, the console needs permission to create *execution roles* for tasks such as `CreateNotebook`, `CreateTrainingJob`, and `CreateModel`. Grant these permissions with the following permissions policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "SageMakerApis",  
            "Effect": "Allow",  
            "Action": [  
                "sagemaker:*"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Sid": "VPCActions",  
            "Effect": "Allow",  
            "Action": [  
                "ec2:DescribeSubnets",  
                "ec2:DescribeVpcs",  
                "ec2:DescribeSecurityGroups"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

```

    "Sid": "VpcConfigurationForCreateForms",
    "Effect": "Allow",
    "Action": [
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups"
    ],
    "Resource": "*"
},
{
    "Sid": "KmsKeysForCreateForms",
    "Effect": "Allow",
    "Action": [
        "kms:DescribeKey",
        "kms>ListAliases"
    ],
    "Resource": "*"
},
{
    "Sid": "AccessAwsMarketplaceSubscriptions",
    "Effect": "Allow",
    "Action": [
        "aws-marketplace:ViewSubscriptions"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "codecommit:BatchGetRepositories",
        "codecommit>CreateRepository",
        "codecommit:GetRepository",
        "codecommit>ListRepositories",
        "codecommit>ListBranches",
        "secretsmanager>CreateSecret",
        "secretsmanager:DescribeSecret",
        "secretsmanager>ListSecrets"
    ],
    "Resource": "*"
},
{
    "Sid": "ListAndCreateExecutionRoles",
    "Effect": "Allow",
    "Action": [
        "iam>ListRoles",
        "iam>CreateRole",
        "iam>CreatePolicy",
        "iam:AttachRolePolicy"
    ],
    "Resource": "*"
},
{
    "Sid": "DescribeECRMetaData",
    "Effect": "Allow",
    "Action": [
        "ecr:Describe*"
    ],
    "Resource": "*"
},
{
    "Sid": "PassRoleForExecutionRoles",
    "Effect": "Allow",
    "Action": [
        "iam:PassRole"
    ],
    "Resource": "*",
}

```

```

        "Condition": {
            "StringEquals": {
                "iam:PassedToService": "sagemaker.amazonaws.com"
            }
        }
    ]
}

```

## Permissions Required to Use the Amazon SageMaker Ground Truth Console

To use the Amazon SageMaker Ground Truth console, you need to grant permissions for additional resources. Specifically, the console needs permissions for the Amazon Marketplace to view subscriptions, Amazon Cognito operations to manage your private workforce, Amazon S3 actions for access to your input and output files, and Amazon Lambda actions to list and invoke functions. Grant these permissions with the following permissions policy:

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GroundTruthConsole",
            "Effect": "Allow",
            "Action": [
                "aws-marketplace:DescribeListings",
                "aws-marketplace:ViewSubscriptions",

                "cognito-idp:AdminAddUserToGroup",
                "cognito-idp:AdminCreateUser",
                "cognito-idp:AdminDeleteUser",
                "cognito-idp:AdminDisableUser",
                "cognito-idp:AdminEnableUser",
                "cognito-idp:AdminRemoveUserFromGroup",
                "cognito-idp>CreateGroup",
                "cognito-idp>CreateUserPool",
                "cognito-idp>CreateUserPoolClient",
                "cognito-idp>CreateUserPoolDomain",
                "cognito-idp:DescribeUserPool",
                "cognito-idp:DescribeUserPoolClient",
                "cognito-idp>ListGroups",
                "cognito-idp>ListIdentityProviders",
                "cognito-idp>ListUsers",
                "cognito-idp>ListUsersInGroup",
                "cognito-idp>ListUserPoolClients",
                "cognito-idp>ListUserPools",
                "cognito-idp:UpdateUserPool",
                "cognito-idp:UpdateUserPoolClient",

                "groundtruthlabeling:DescribeConsoleJob",
                "groundtruthlabeling>ListDatasetObjects",
                "groundtruthlabeling:RunFilterOrSampleManifestJob",
                "groundtruthlabeling:RunGenerateManifestByCrawlingJob",

                "lambda:InvokeFunction",
                "lambda>ListFunctions",

                "s3:GetObject",
                "s3:PutObject",
                "s3>SelectObjectContent"
            ],
            "Resource": "*"
        }
    ]
}

```

}

## Permissions Required to Use the Amazon Augmented AI (Preview) Console

To use the Augmented AI console, you need to grant permissions for additional resources. Grant these permissions with the following permissions policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sagemaker:*Algorithm",  
                "sagemaker:*Algorithms",  
                "sagemaker:*App",  
                "sagemaker:*Apps",  
                "sagemaker:*AutoMLJob",  
                "sagemaker:*AutoMLJobs",  
                "sagemaker:*CodeRepositories",  
                "sagemaker:*CodeRepository",  
                "sagemaker:*CompilationJob",  
                "sagemaker:*CompilationJobs",  
                "sagemaker:*Endpoint",  
                "sagemaker:*EndpointConfig",  
                "sagemaker:*EndpointConfigs",  
                "sagemaker:*EndpointWeightsAndCapacities",  
                "sagemaker:*Endpoints",  
                "sagemaker:*Environment",  
                "sagemaker:*EnvironmentVersion",  
                "sagemaker:*EnvironmentVersions",  
                "sagemaker:*Environments",  
                "sagemaker:*Experiment",  
                "sagemaker:*Experiments",  
                "sagemaker:*FlowDefinitions",  
                "sagemaker:*HumanLoop",  
                "sagemaker:*HumanLoops",  
                "sagemaker:*HumanTaskUi",  
                "sagemaker:*HumanTaskUis",  
                "sagemaker:*HyperParameterTuningJob",  
                "sagemaker:*HyperParameterTuningJobs",  
                "sagemaker:*LabelingJob",  
                "sagemaker:*LabelingJobs",  
                "sagemaker:*Metrics",  
                "sagemaker:*Model",  
                "sagemaker:*ModelPackage",  
                "sagemaker:*ModelPackages",  
                "sagemaker:*Models",  
                "sagemaker:*MonitoringExecutions",  
                "sagemaker:*MonitoringSchedule",  
                "sagemaker:*MonitoringSchedules",  
                "sagemaker:*NotebookInstance",  
                "sagemaker:*NotebookInstanceLifecycleConfig",  
                "sagemaker:*NotebookInstanceLifecycleConfigs",  
                "sagemaker:*NotebookInstanceUrl",  
                "sagemaker:*NotebookInstances",  
                "sagemaker:*ProcessingJob",  
                "sagemaker:*ProcessingJobs",  
                "sagemaker:*RenderUiTemplate",  
                "sagemaker:*Search",  
                "sagemaker:*SearchSuggestions",  
                "sagemaker:*Tags",  
                "sagemaker:*TrainingJob",  
                "sagemaker:*TrainingJobs",  
            ]  
        }  
    ]  
}
```

```

        "sagemaker:*TransformJob",
        "sagemaker:*TransformJobs",
        "sagemaker:*Trial",
        "sagemaker:*TrialComponent",
        "sagemaker:*TrialComponents",
        "sagemaker:*Trials",
        "sagemaker:*Workteam",
        "sagemaker:*Workteams"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "sagemaker:*FlowDefinition"
    ],
    "Resource": "*",
    "Condition": {
        "StringEqualsIfExists": {
            "sagemaker:WorkteamType": [
                "private-crowd",
                "vendor-crowd"
            ]
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "application-autoscaling>DeleteScalingPolicy",
        "application-autoscaling>DeleteScheduledAction",
        "application-autoscaling>DeregisterScalableTarget",
        "application-autoscaling>DescribeScalableTargets",
        "application-autoscaling>DescribeScalingActivities",
        "application-autoscaling>DescribeScalingPolicies",
        "application-autoscaling>DescribeScheduledActions",
        "application-autoscaling>PutScalingPolicy",
        "application-autoscaling>PutScheduledAction",
        "application-autoscaling>RegisterScalableTarget",
        "aws-marketplace>ViewSubscriptions",
        "cloudwatch>DeleteAlarms",
        "cloudwatch>DescribeAlarms",
        "cloudwatch>GetMetricData",
        "cloudwatch>GetMetricStatistics",
        "cloudwatch>ListMetrics",
        "cloudwatch>PutMetricAlarm",
        "cloudwatch>PutMetricData",
        "codecommit>BatchGetRepositories",
        "codecommit>CreateRepository",
        "codecommit>GetRepository",
        "codecommit>ListBranches",
        "codecommit>ListRepositories",
        "cognito-idp:AdminAddUserToGroup",
        "cognito-idp:AdminCreateUser",
        "cognito-idp:AdminDeleteUser",
        "cognito-idp:AdminDisableUser",
        "cognito-idp:AdminEnableUser",
        "cognito-idp:AdminRemoveUserFromGroup",
        "cognito-idp>CreateGroup",
        "cognito-idp>CreateUserPool",
        "cognito-idp>CreateUserPoolClient",
        "cognito-idp>CreateUserPoolDomain",
        "cognito-idp>DescribeUserPool",
        "cognito-idp>DescribeUserPoolClient",
        "cognito-idp>ListGroups",
        "cognito-idp>ListIdentityProviders",

```

```

    "cognito-idp>ListUserPoolClients",
    "cognito-idp>ListUserPools",
    "cognito-idp>ListUsers",
    "cognito-idp>ListUsersInGroup",
    "cognito-idp>UpdateUserPool",
    "cognito-idp>UpdateUserPoolClient",
    "ec2>CreateNetworkInterface",
    "ec2>CreateNetworkInterfacePermission",
    "ec2>CreateVpcEndpoint",
    "ec2>DeleteNetworkInterface",
    "ec2>DeleteNetworkInterfacePermission",
    "ec2>DescribeDhcpOptions",
    "ec2>DescribeNetworkInterfaces",
    "ec2>DescribeRouteTables",
    "ec2>DescribeSecurityGroups",
    "ec2>DescribeSubnets",
    "ec2>DescribeVpcEndpoints",
    "ec2>DescribeVpcs",
    "ecr>BatchCheckLayerAvailability",
    "ecr>BatchGetImage",
    "ecr>CreateRepository",
    "ecr>Describe*",
    "ecr>GetAuthorizationToken",
    "ecr>GetDownloadUrlForLayer",
    "elastic-inference:Connect",
    "elasticfilesystem:DescribeFileSystems",
    "elasticfilesystem:DescribeMountTargets",
    "fsx:DescribeFileSystems",
    "glue>CreateJob",
    "glue>DeleteJob",
    "glue:GetJob",
    "glue:GetJobRun",
    "glue:GetJobRuns",
    "glue:GetJobs",
    "glue:ResetJobBookmark",
    "glue:StartJobRun",
    "glue:UpdateJob",
    "groundtruthlabeling:*",
    "iam>ListRoles",
    "kms>DescribeKey",
    "kms>ListAliases",
    "lambda>ListFunctions",
    "logs>CreateLogGroup",
    "logs>CreateLogStream",
    "logs>DescribeLogGroups",
    "logs>DescribeLogStreams",
    "logs>GetLogEvents",
    "logs>PutLogEvents",
    "sns>ListTopics"
],
"Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "logs>CreateLogDelivery",
        "logs>DeleteLogDelivery",
        "logs>DescribeResourcePolicies",
        "logs>GetLogDelivery",
        "logs>ListLogDeliveries",
        "logs>PutResourcePolicy",
        "logs>UpdateLogDelivery"
    ],
    "Resource": "*"
}
{

```

```

    "Effect": "Allow",
    "Action": [
        "ecr:SetRepositoryPolicy",
        "ecr:CompleteLayerUpload",
        "ecr:BatchDeleteImage",
        "ecr:UploadLayerPart",
        "ecr:DeleteRepositoryPolicy",
        "ecr:InitiateLayerUpload",
        "ecr:DeleteRepository",
        "ecr:PutImage"
    ],
    "Resource": "arn:aws:ecr:*::repository/*sagemaker*"
},
{
    "Effect": "Allow",
    "Action": [
        "codecommit:GitPull",
        "codecommit:GitPush"
    ],
    "Resource": [
        "arn:aws:codecommit:*::sagemaker*",
        "arn:aws:codecommit:*::SageMaker*",
        "arn:aws:codecommit:*::Sagemaker*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "secretsmanager>ListSecrets"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "secretsmanager>DescribeSecret",
        "secretsmanager>GetSecretValue",
        "secretsmanager>CreateSecret"
    ],
    "Resource": [
        "arn:aws:secretsmanager:*::secret:AmazonSageMaker-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "secretsmanager>DescribeSecret",
        "secretsmanager>GetSecretValue"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "secretsmanager:ResourceTag/SageMaker": "true"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "robomaker>CreateSimulationApplication",
        "robomaker>DescribeSimulationApplication",
        "robomaker>DeleteSimulationApplication"
    ],
    "Resource": [
        "*"
    ]
}

```

```

},
{
    "Effect": "Allow",
    "Action": [
        "robomaker>CreateSimulationJob",
        "robomaker>DescribeSimulationJob",
        "robomaker>CancelSimulationJob"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:AbortMultipartUpload",
        "s3:GetBucketCors",
        "s3:PutBucketCors"
    ],
    "Resource": [
        "arn:aws:s3::::*SageMaker*",
        "arn:aws:s3::::*Sagemaker*",
        "arn:aws:s3::::*sagemaker*",
        "arn:aws:s3::::aws-glue*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3>CreateBucket",
        "s3:GetBucketLocation",
        "s3>ListBucket",
        "s3>ListAllMyBuckets"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject"
    ],
    "Resource": "*",
    "Condition": {
        "StringEqualsIgnoreCase": {
            "s3:ExistingObjectTag/SageMaker": "true"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "lambda:InvokeFunction"
    ],
    "Resource": [
        "arn:aws:lambda::::function:*SageMaker*",
        "arn:aws:lambda::::function:*sagemaker*",
        "arn:aws:lambda::::function:*Sagemaker*",
        "arn:aws:lambda::::function:*LabelingFunction*"
    ]
},
{
    "Action": "iam>CreateServiceLinkedRole",
    "Effect": "Allow",

```

```

    "Resource": "arn:aws:iam::*:role/aws-service-role/sagemaker.application-
autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "sagemaker.application-autoscaling.amazonaws.com"
        }
    }
},
{
    "Effect": "Allow",
    "Action": "iam:CreateServiceLinkedRole",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "iam:AWSServiceName": "robomaker.amazonaws.com"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "sns:Subscribe",
        "sns:CreateTopic"
    ],
    "Resource": [
        "arn:aws:sns:***:*SageMaker*",
        "arn:aws:sns:***:*Sagemaker*",
        "arn:aws:sns:***:*sagemaker*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "iam:PassRole"
    ],
    "Resource": "arn:aws:iam::*:role/*",
    "Condition": {
        "StringEquals": {
            "iam:PassedToService": [
                "sagemaker.amazonaws.com",
                "glue.amazonaws.com",
                "robomaker.amazonaws.com",
                "states.amazonaws.com"
            ]
        }
    }
}
]
}

```

## Allow Users to View Their Own Permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the Amazon CLI or Amazon API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",

```

```
        "iam>ListGroupsForUser",
        "iam>ListAttachedUserPolicies",
        "iam>ListUserPolicies",
        "iam GetUser"
    ],
    "Resource": ["arn:aws-cn:iam::*:user/${aws:username}"]
},
{
    "Sid": "NavigateInConsole",
    "Effect": "Allow",
    "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam>ListAttachedGroupPolicies",
        "iam>ListGroupPolicies",
        "iam>ListPolicyVersions",
        "iam>ListPolicies",
        "iam>ListUsers"
    ],
    "Resource": "*"
}
]
```

## Control Creation of SageMaker Resources with Condition Keys

Control fine-grained access to allow the creation of SageMaker resources by using SageMaker-specific condition keys. For information about using condition keys in IAM policies, see [IAM JSON Policy Elements: Condition](#) in the *IAM User Guide*.

The condition keys, along with related API actions, and links to relevant documentation are listed in [Condition Keys for SageMaker](#) in the *IAM User Guide*.

The following examples show how to use the SageMaker condition keys to control access.

### Topics

- [Control Access to SageMaker Resources by Using File System Condition Keys \(p. 2434\)](#)
- [Restrict Training to a Specific VPC \(p. 2436\)](#)
- [Restrict Access to Workforce Types for Ground Truth Labeling Jobs and Amazon A2I Human Review Workflows \(p. 2437\)](#)
- [Enforce Encryption of Input Data \(p. 2438\)](#)
- [Enforce Encryption of Notebook Instance Storage Volume \(p. 2438\)](#)
- [Enforce Network Isolation for Training Jobs \(p. 2439\)](#)
- [Enforce a Specific Instance Type for Training Jobs \(p. 2439\)](#)
- [Enforce a Specific EI Accelerator for Training Jobs \(p. 2440\)](#)
- [Enforce Disabling Internet Access and Root Access for Creating Notebook Instances \(p. 2440\)](#)

## Control Access to SageMaker Resources by Using File System Condition Keys

SageMaker training provides a secure infrastructure for the training algorithm to run in, but for some cases you may want increased defense in depth. For example, you minimize the risk of running untrusted code in your algorithm, or you have specific security mandates in your organization. For these scenarios, you can use the service-specific condition keys in the Condition element of an IAM policy to scope down the user to specific file systems, directories, access modes (read-write, read-only) and security groups.

### Topics

- [Restrict an IAM User to Specific Directories and Access Modes \(p. 2435\)](#)
- [Restrict an IAM User to a Specific File System \(p. 2435\)](#)

### [Restrict an IAM User to Specific Directories and Access Modes](#)

The policy below restricts an IAM user to the `/sagemaker/xgboost-dm/train` and `/sagemaker/xgboost-dm/validation` directories of an EFS file system to `ro` (read-only) AccessMode:

**Note**

When a directory is allowed, all of its subdirectories are also accessible by the training algorithm. POSIX permissions are ignored.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AccessToElasticFileSystem",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:FileSystemId": "fs-12345678",
                    "sagemaker:FileSystemAccessMode": "ro",
                    "sagemaker:FileSystemType": "EFS",
                    "sagemaker:FileSystemDirectoryPath": "/sagemaker/xgboost-dm/train"
                }
            }
        },
        {
            "Sid": "AccessToElasticFileSystemValidation",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:FileSystemId": "fs-12345678",
                    "sagemaker:FileSystemAccessMode": "ro",
                    "sagemaker:FileSystemType": "EFS",
                    "sagemaker:FileSystemDirectoryPath": "/sagemaker/xgboost-dm/validation"
                }
            }
        }
    ]
}
```

### [Restrict an IAM User to a Specific File System](#)

To prevent a malicious algorithm using a user space client from accessing any file system directly in your account, you can restrict networking traffic by allowing ingress from a specific security group. In the following example, the IAM user can only use the specified security group to access the file system:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {

```

```
        "Sid": "AccessToLustreFileSystem",
        "Effect": "Allow",
        "Action": [
            "sagemaker>CreateTrainingJob",
            "sagemaker>CreateHyperParameterTuningJob"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "sagemaker:FileSystemId": "fs-12345678",
                "sagemaker:FileSystemAccessMode": "ro",
                "sagemaker:FileSystemType": "FSxLustre",
                "sagemaker:FileSystemDirectoryPath": "/fsx/sagemaker/xgboost/train"
            },
            "ForAllValues:StringEquals": {
                "sagemaker:VpcSecurityGroupIds": [
                    "sg-12345678"
                ]
            }
        }
    }
]
```

Although the above example can restrict an algorithm to a specific file system, it does not prevent an algorithm from accessing any directory within that file system using the user space client. To mitigate this, you can:

- Ensure that the file system only contains data that you trust your IAM users to access
  - Create an IAM role that restricts your IAM users to launching training jobs with algorithms from approved ECR repositories

For more information on how to use roles with SageMaker, see [SageMaker Roles](#).

## Restrict Training to a Specific VPC

Restrict an Amazon user to creating training jobs from within a Amazon VPC. When a training job is created within a VPC, you can use VPC flow logs to monitor all traffic to and from the training cluster. For information about using VPC flow logs, see [VPC Flow Logs](#) in the *Amazon Virtual Private Cloud User Guide*.

The following policy enforces that a training job is created by an IAM user calling `CreateTrainingJob` from within a VPC:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowFromVpc",  
            "Effect": "Allow",  
            "Action": [  
                "sagemaker:CreateTrainingJob",  
                "sagemaker:CreateHyperParameterTuningJob"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "ForAllValues:StringEquals": {  
                    "sagemaker:VpcSubnets": ["subnet-a1234"],  
                    "sagemaker:VpcSecurityGroupIds": ["sg12345", "sg-67890"]  
                },  
                "Null": {  
                    "aws:SourceVpc": "  
                }  
            }  
        }  
    ]  
}
```

```

        "sagemaker:VpcSubnets": "false",
        "sagemaker:VpcSecurityGroupIds": "false"
    }
}
]
}
```

## Restrict Access to Workforce Types for Ground Truth Labeling Jobs and Amazon A2I Human Review Workflows

Amazon SageMaker Ground Truth and Amazon Augmented AI work teams fall into one of three [workforce types](#): public (with Amazon Mechanical Turk), private, and vendor. To restrict IAM user access to a specific work team using one of these types or the work team ARN, use the `sagemaker:WorkteamType` and/or the `sagemaker:WorkteamArn` condition keys. For the `sagemaker:WorkteamType` condition key, use [string condition operators](#). For the `sagemaker:WorkteamArn` condition key, use [Amazon Resource Name \(ARN\) condition operators](#). If the user attempts to create a labeling job with a restricted work team, SageMaker returns an access denied error.

The policies below demonstrate different ways to use the `sagemaker:WorkteamType` and `sagemaker:WorkteamArn` condition keys with appropriate condition operators and valid condition values.

The following example uses the `sagemaker:WorkteamType` condition key with the `StringEquals` condition operator to restrict access to a public work team. It accepts condition values in the following format: `workforcetype-crowd`, where `workforcetype` can equal `public`, `private`, or `vendor`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RestrictWorkteamType",
            "Effect": "Deny",
            "Action": "sagemaker>CreateLabelingJob",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:WorkteamType": "public-crowd"
                }
            }
        }
    ]
}
```

The following policies show how to restrict access to a public work team using the `sagemaker:WorkteamArn` condition key. The first shows how to use it with a valid IAM regex-variant of the work team ARN and the `ArnLike` condition operator. The second shows how to use it with the `ArnEquals` condition operator and the work team ARN.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RestrictWorkteamType",
            "Effect": "Deny",
            "Action": "sagemaker>CreateLabelingJob",
            "Resource": "*",
            "Condition": {
                "ArnLike": {

```

```
*"
        }
    }
}
]

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "RestrictWorkteamType",
            "Effect": "Deny",
            "Action": "sagemaker>CreateLabelingJob",
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "sagemaker:WorkteamArn": "arn:aws:sagemaker:us-west-2:394669845002:workteam/public-crowd/default"
                }
            }
        }
    ]
}
```

## Enforce Encryption of Input Data

The following policy restricts an IAM user to specify a Amazon KMS key to encrypt input data when creating training, hyperparameter tuning, and labeling jobs by using the `sagemaker:VolumeKmsKey` condition key:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceEncryption",
            "Effect": "Allow",
            "Action": [
                "sagemaker>CreateTrainingJob",
                "sagemaker>CreateHyperParameterTuningJob",
                "sagemaker>CreateLabelingJob",
                "sagemaker>CreateFlowDefiniton"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "sagemaker:VolumeKmsKey": "arn:aws:kms:us-west-2:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab"
                }
            }
        }
    ]
}
```

## Enforce Encryption of Notebook Instance Storage Volume

The following policy restricts an IAM user to specify a Amazon KMS key to encrypt the attached storage volume when creating or updating a notebook instance by using the `sagemaker:VolumeKmsKey` condition key:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceEncryption",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateNotebookInstance"
            ],
            "Resource": "*",
            "Condition": {
                "ArnLike": {
                    "sagemaker:VolumeKmsKey": "*key/volume-kms-key-12345"
                }
            }
        }
    ]
}
```

## Enforce Network Isolation for Training Jobs

The following policy restricts an IAM user to enable network isolation when creating training jobs by using the `sagemaker:NetworkIsolation` condition key:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceIsolation",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "Bool": {
                    "sagemaker:NetworkIsolation": "true"
                }
            }
        }
    ]
}
```

## Enforce a Specific Instance Type for Training Jobs

The following policy restricts an IAM user to use a specific instance type when creating training jobs by using the `sagemaker:InstanceTypes` condition key:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceInstanceType",
            "Effect": "Allow",
            "Action": [
                "sagemaker:CreateTrainingJob",
                "sagemaker:CreateHyperParameterTuningJob"
            ],
            "Resource": "*",
            "Condition": {
                "String": {
                    "sagemaker:InstanceTypes": "ml.m5.xlarge"
                }
            }
        }
    ]
}
```

```

        "Condition": {
            "ForAllValues:StringLike": {
                "sagemaker:InstanceTypes": ["ml.c5.*"]
            }
        }
    ]
}

```

## Enforce a Specific EI Accelerator for Training Jobs

The following policy restricts an IAM user to use a specific elastic inference (EI) accelerator, if an accelerator is provided, when creating or updating notebook instances and when creating endpoint configurations by using the `sagemaker:AcceleratorTypes` condition key:

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceAcceleratorType",
            "Effect": "Allow",
            "Action": [
                "sagemaker>CreateNotebookInstance",
                "sagemaker:UpdateNotebookInstance",
                "sagemaker>CreateEndpointConfig"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "sagemaker:AcceleratorTypes": ["ml.eia1.medium"]
                }
            }
        }
    ]
}

```

## Enforce Disabling Internet Access and Root Access for Creating Notebook Instances

You can disable both internet access and root access to notebook instances to help make them more secure. For information about controlling root access to a notebook instance, see [Control root access to a SageMaker notebook instance \(p. 2412\)](#). For information about disabling internet access for a notebook instance, see [Connect a Notebook Instance to Resources in a VPC \(p. 2493\)](#).

The following policy requires an IAM user to disable network access when creating instance, and disable root access when creating or updating a notebook instance.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LockDownCreateNotebookInstance",
            "Effect": "Allow",
            "Action": [
                "sagemaker>CreateNotebookInstance"
            ],
            "Resource": "*",
            "Condition": {

```

```

        "StringEquals": {
            "sagemaker:DirectInternetAccess": "Disabled",
            "sagemaker:RootAccess": "Disabled"
        },
        "Null": {
            "sagemaker:VpcSubnets": "false",
            "sagemaker:VpcSecurityGroupIds": "false"
        }
    }
}

{
    "Sid": "LockDownUpdateNotebookInstance",
    "Effect": "Allow",
    "Action": [
        "sagemaker:UpdateNotebookInstance"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "sagemaker:RootAccess": "Disabled"
        }
    }
}
]
}

```

## Control Access to the SageMaker API by Using Identity-based Policies

To control access to SageMaker API calls and calls to SageMaker hosted endpoints, use identity-based IAM policies.

### Topics

- [Restrict Access to SageMaker API and Runtime to Calls from Within Your VPC \(p. 2441\)](#)

## Restrict Access to SageMaker API and Runtime to Calls from Within Your VPC

If you set up an interface endpoint in your VPC, individuals outside the VPC can still connect to the SageMaker API and runtime over the internet unless you attach an IAM policy that restricts access to calls coming from within the VPC to all users and groups that have access to your SageMaker resources. For information about creating a VPC interface endpoint for the SageMaker API and runtime, see [Connect to SageMaker Through a VPC Interface Endpoint \(p. 2495\)](#).

### Important

If you apply an IAM policy similar to one of the following, users can't access the specified SageMaker APIs through the console.

To restrict access to only connections made from within your VPC, create an Amazon Identity and Access Management policy that restricts access to only calls that come from within your VPC. Then add that policy to every Amazon Identity and Access Management user, group, or role used to access the SageMaker API or runtime.

### Note

This policy allows connections only to callers within a subnet where you created an interface endpoint.

```
{
    "Id": "api-example-1",
    "Version": "2012-10-17",
}
```

```

"Statement": [
    {
        "Sid": "Enable API Access",
        "Effect": "Allow",
        "Action": [
            "sagemaker:*"
        ],
        "Resource": "*",
        "Condition": {
            "StringEquals": {
                "aws:SourceVpc": "vpc-111bbaaa"
            }
        }
    }
]
}

```

If you want to restrict access to the API to only calls made using the interface endpoint, use the `aws:SourceVpce` condition key instead of `aws:SourceVpc`:

```

{
    "Id": "api-example-1",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable API Access",
            "Effect": "Allow",
            "Action": [
                "sagemaker>CreatePresignedNotebookInstanceUrl"
            ],
            "Resource": "*",
            "Condition": {
                "ForAllValues:StringEquals": {
                    "aws:sourceVpce": [
                        "vpce-111bbccc",
                        "vpce-111bbddd"
                    ]
                }
            }
        }
    ]
}

```

## Limit Access to SageMaker API and Runtime Calls by IP Address

To allow access to SageMaker API calls and runtime invocations only from IP addresses in a list that you specify, attach an IAM policy that denies access to the API unless the call comes from an IP address in the list to every Amazon Identity and Access Management user, group, or role used to access the API or runtime. For information about creating IAM policies, see [Creating IAM Policies](#) in the *Amazon Identity and Access Management User Guide*. To specify the list of IP addresses that you want to have access to the API call, use the `IpAddress` condition operator and the `aws:SourceIP` condition context key. For information about IAM condition operators, see [IAM JSON Policy Elements: Condition Operators](#) in the *Amazon Identity and Access Management User Guide*. For information about IAM condition context keys, see [Amazon Global Condition Context Keys](#).

For example, the following policy allows access to the `CreateTrainingJob` only from IP addresses in the ranges 192.0.2.0-192.0.2.255 and 203.0.113.0-203.0.113.255:

```

{
    "Version": "2012-10-17",

```

```

"Statement": [
    {
        "Effect": "Allow",
        "Action": "sagemaker>CreateTrainingJob",
        "Resource": "*",
        "Condition": {
            "IpAddress": {
                "aws:SourceIp": [
                    "192.0.2.0/24",
                    "203.0.113.0/24"
                ]
            }
        }
    }
]
}

```

## Limit Access to a Notebook Instance by IP Address

To allow access to a notebook instance only from IP addresses in a list that you specify, attach an IAM policy that denies access to `CreatePresignedNotebookInstanceUrl` unless the call comes from an IP address in the list to every Amazon Identity and Access Management user, group, or role used to access the notebook instance. For information about creating IAM policies, see [Creating IAM Policies](#) in the *Amazon Identity and Access Management User Guide*. To specify the list of IP addresses that you want to have access to the notebook instance, use the `IpAddress` condition operator and the `aws:SourceIP` condition context key. For information about IAM condition operators, see [IAM JSON Policy Elements: Condition Operators](#) in the *Amazon Identity and Access Management User Guide*. For information about IAM condition context keys, see [Amazon Global Condition Context Keys](#).

For example, the following policy allows access to a notebook instance only from IP addresses in the ranges 192.0.2.0-192.0.2.255 and 203.0.113.0-203.0.113.255:

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sagemaker>CreatePresignedNotebookInstanceUrl",
            "Resource": "*",
            "Condition": {
                "IpAddress": {
                    "aws:SourceIp": [
                        "192.0.2.0/24",
                        "203.0.113.0/24"
                    ]
                }
            }
        }
    ]
}

```

The policy restricts access to both the call to `CreatePresignedNotebookInstanceUrl` and to the URL that the call returns. The policy also restricts access to opening a notebook instance in the console and is enforced for every HTTP request and WebSocket frame that attempts to connect to the notebook instance.

**Note**

Using this method to filter by IP address is incompatible when [connecting to SageMaker through a VPC interface endpoint](#). For information about restricting access to a notebook

instance when connecting through a VPC interface endpoint, see [Connect to a Notebook Instance Through a VPC Interface Endpoint \(p. 2497\)](#).

## Control Access to SageMaker Resources by Using Tags

Control access to groups of SageMaker resources by attaching tags to the resources and specifying `ResourceTag` conditions in IAM policies.

### Note

Tag-based policies don't work to restrict the following API calls:

- `ListAlgorithms`
- `ListCodeRepositories`
- `ListCompilationJobs`
- `ListEndpointConfigs`
- `ListEndpoints`
- `ListFlowDefinitions`
- `ListHumanTaskUis`
- `ListHyperparameterTuningJobs`
- `ListLabelingJobs`
- `ListLabelingJobsForWorkteam`
- `ListModelPackages`
- `ListModels`
- `ListNotebookInstanceLifecycleConfigs`
- `ListNotebookInstances`
- `ListSubscribedWorkteams`
- `ListTags`
- `ListProcessingJobs`
- `ListTrainingJobs`
- `ListTrainingJobsForHyperParameterTuningJob`
- `ListTransformJobs`
- `ListWorkteams`
- `Search`

For example, suppose you've defined two different IAM groups, named `DevTeam1` and `DevTeam2`, in your Amazon account. Suppose also that you've created 10 notebook instances, 5 of which are used for one project, and 5 of which are used for a second project. You want to allow members of `DevTeam1` to make API calls on notebook instances used for the first project, and members of `DevTeam2` to make API calls on notebook instances used for the second project.

### To control access to API calls (example)

1. Add a tag with the key `Project` and value `A` to the notebook instances used for the first project. For information about adding tags to SageMaker resources, see [AddTags](#).
2. Add a tag with the key `Project` and value `B` to the notebook instances used for the second project.
3. Create an IAM policy with a `ResourceTag` condition that denies access to the notebook instances used for the second project, and attach that policy to `DevTeam1`. The following is an example of a policy that denies all API calls on any notebook instance that has a tag with a key of `Project` and a value of `B`:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sagemaker:*",
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": "sagemaker:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:ResourceTag/Project": "B"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
                "sagemaker:AddTags",
                "sagemaker:DeleteTags"
            ],
            "Resource": "*"
        }
    ]
}
```

For information about creating IAM policies and attaching them to identities, see [Controlling Access Using Policies](#) in the *Amazon Identity and Access Management User Guide*.

4. Create an IAM policy with a ResourceTag condition that denies access to the notebook instances used for the first project, and attach that policy to DevTeam2. The following is an example of a policy that denies all API calls on any notebook instance that has a tag with a key of Project and a value of A:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "*",
            "Resource": "*"
        },
        {
            "Effect": "Deny",
            "Action": "sagemaker:*",
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "sagemaker:ResourceTag/Project": "A"
                }
            }
        },
        {
            "Effect": "Deny",
            "Action": [
                "sagemaker:AddTags",
                "sagemaker:DeleteTags"
            ],
            "Resource": "*"
        }
    ]
}
```

```
    ]  
}
```

## Require the Presence or Absence of Tags for API Calls

Require the presence or absence of specific tags or specific tag values by using `RequestTag` condition keys in an IAM policy. For example, if you want to require that every endpoint created by any member of an IAM group to be created with a tag with the key `environment` and value `dev`, create a policy as follows:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    },
    {
      "Effect": "Deny",
      "Action": "sagemaker>CreateEndpoint",
      "Resource": "arn:aws:sagemaker:*::endpoint/*",
      "Condition": {
        "StringNotEquals": {
          "aws:RequestTag/environment": "dev"
        }
      }
    }
  ]
}
```

## Use Tags with Hyperparameter Tuning Jobs

You can add tags to a hyperparameter tuning job when you create the tuning job by specifying the tags as the `Tags` parameter when you call [CreateHyperParameterTuningJob](#). If you do this, the tags you specify for the hyperparameter tuning job are also added to all training jobs that the hyperparameter tuning job launches.

If you add tags to a hyperparameter tuning job by calling [AddTags](#), the tags you add are also added to any training jobs that the hyperparameter tuning job launches after you call [AddTags](#), but are not added to training jobs the hyperparameter tuning jobs launched before you called [AddTags](#). Similarly, when you remove tags from a hyperparameter tuning job by calling [DeleteTags](#), those tags are not removed from training jobs that the hyperparameter tuning job launched previously. Because of this, the tags associated with training jobs can be out of sync with the tags associated with the hyperparameter tuning job that launched them. If you use tags to control access to a hyperparameter tuning job and the training jobs it launches, you might want to keep the tags in sync. To make sure the tags associated with training jobs stay sync with the tags associated with the hyperparameter tuning job that launched them, first call [ListTrainingJobsForHyperParameterTuningJob](#) for the hyperparameter tuning job to get a list of the training jobs that the hyperparameter tuning job launched. Then, call [AddTags](#) or [DeleteTags](#) for the hyperparameter tuning job and for each of the training jobs in the list of training jobs to add or delete the same set of tags for all of the jobs. The following Python example demonstrates this:

```
tuning_job_arn =
smclient.describe_hyper_parameter_tuning_job(HyperParameterTuningJobName='MyTuningJob')
['HyperParameterTuningJobArn']
smclient.add_tags(ResourceArn=tuning_job_arn, Tags=[{'Key': 'Env', 'Value': 'Dev'}])
training_jobs = smclient.list_training_jobs_for_hyper_parameter_tuning_job(
    HyperParameterTuningJobName='MyTuningJob')['TrainingJobSummaries']
for training_job in training_jobs:
```

```
time.sleep(1) # Wait for 1 second between calls to avoid being throttled
smclient.add_tags(ResourceArn=training_job['TrainingJobArn'], Tags=[{'Key': 'Env',
'Value':'Dev'}])
```

## SageMaker Roles

As a managed service, SageMaker performs operations on your behalf on the Amazon hardware that is managed by SageMaker. SageMaker can perform only operations that the user permits.

A SageMaker user can grant these permissions with an IAM role (referred to as an execution role).

To create and use a locally available execution role, you can use the following procedures.

### Get execution role

When you run a notebook within SageMaker you can access the execution role with the following code:

```
sagemaker_session = sagemaker.Session()
role = sagemaker.get_execution_role()
```

#### Note

The execution role is intended to be available only when running a notebook within SageMaker. If you run `get_execution_role` in a notebook not on SageMaker, expect a "region" error.

To find the IAM role ARN created when you created your the notebook instance or Studio application, go to the **Notebook instances** page in the console and select the relevant notebook from the list of **Names**. In the configuration detail page the IAM role ARN is given in the **Permissions and encryption** section.

Use the following procedure to create an execution role with the IAM managed policy, `AmazonSageMakerFullAccess`, attached. If your use case requires more granular permissions, use other sections on this page to create an execution role that meets your business needs.

#### Important

The IAM managed policy, `AmazonSageMakerFullAccess`, used in the following procedure only grants the execution role permission to perform certain Amazon S3 actions on buckets or objects with SageMaker, Sagemaker, sagemaker, or aws-glue in the name. To learn how to add an additional policy to an execution role to grant it access to other Amazon S3 buckets and objects, see [Add Additional Amazon S3 Permissions to an SageMaker Execution Role \(p. 2448\)](#).

### To create a new role

1. Open the IAM console at <https://console.amazonaws.cn/iam/>.
2. Select **Roles** and then select **Create role**.
3. Select **SageMaker**.
4. Select **Next: Permissions**.
5. The IAM managed policy, `AmazonSageMakerFullAccess` is automatically attached to this role. To see the permissions included in this policy, select the sideways arrow next to the policy name. Select **Next: Tags**.
6. (Optional) Add tags and select **Next: Review**.
7. Give the role a name in the text field under **Role name** and select **Create role**.
8. On the **Roles** section of the IAM console, select the role you just created. If needed, use the text box to search for the role using the role name you entered in step 7.
9. On the role summary page, make note of the ARN.

With a known ARN for your role, you can programmatically check the role when running the notebook locally or on SageMaker. Replace `RoleName` with your known ARN:

```

try:
    role = sagemaker.get_execution_role()
except ValueError:
    iam = boto3.client('iam')
    role = iam.get_role(RoleName='AmazonSageMaker-ExecutionRole-20201200T100000')['Role']
    ['Arn']

```

## Add Additional Amazon S3 Permissions to an SageMaker Execution Role

When you use an SageMaker feature with resources in Amazon S3, such as input data, the execution role you specify in your request (for example `CreateTrainingJob`) is used to access these resources.

If you attach the IAM managed policy, `AmazonSageMakerFullAccess`, to an execution role, that role has permission to perform certain Amazon S3 actions on buckets or objects with `SageMaker`, `Sagemaker`, `sagemaker`, or `aws-glue` in the name. It also has permission to perform the following actions on any Amazon S3 resource:

```

"s3:CreateBucket",
"s3:GetBucketLocation",
"s3>ListBucket",
"s3>ListAllMyBuckets",
"s3:GetBucketCors",
"s3:PutBucketCors"

```

To give an execution role permissions to access one or more specific buckets in Amazon S3, you can attach a policy similar to the following to the role. This policy grants an IAM role permission to perform all actions that `AmazonSageMakerFullAccess` allows but restricts this access to the buckets and . Refer to the security documentation for the specific SageMaker feature you are using to learn more about the Amazon S3 permissions required for that feature.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3>DeleteObject",
                "s3:AbortMultipartUpload"
            ],
            "Resource": [
                "arn:aws:s3:::/*",
                "arn:aws:s3:::/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:GetBucketLocation",
                "s3>ListBucket",
                "s3>ListAllMyBuckets",
                "s3:GetBucketCors",
                "s3:PutBucketCors"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [

```

```
        "s3:GetBucketAcl",
        "s3:PutObjectAcl"
    ],
    "Resource": [
        "arn:aws:s3:::",
        "arn:aws:s3:::"
    ]
}
}
```

## Passing Roles

Actions like passing a role between services are a common function within SageMaker. You can find more details on [Actions, Resources, and Condition Keys for SageMaker](#) in the *IAM User Guide*.

You pass the role (`iam:PassRole`) when making these API calls: [CreateAutoMLJob](#), [CreateCompilationJob](#), [CreateDomain](#), [CreateFlowDefiniton](#), [CreateHyperParameterTuningJob](#), [CreateImage](#), [CreateLabelingJob](#), [CreateModel](#), [CreateMonitoringSchedule](#), [CreateNotebookInstance](#), [CreateProcessingJob](#), [CreateTrainingJob](#), [CreateUserProfile](#), [RenderUiTemplate](#), and [UpdateImage](#).

You attach the following trust policy to the IAM role which grants SageMaker principal permissions to assume the role, and is the same for all of the execution roles:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "sagemaker.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

The permissions that you need to grant to the role vary depending on the API that you call. The following sections explain these permissions.

### Note

Instead of managing permissions by crafting a permission policy, you can use the Amazon-managed [AmazonSageMakerFullAccess](#) permission policy. The permissions in this policy are fairly broad, to allow for any actions you might want to perform in SageMaker. For a listing of the policy including information about the reasons for adding many of the permissions, see [AmazonSageMakerFullAccess \(p. 2465\)](#). If you prefer to create custom policies and manage permissions to scope the permissions only to the actions you need to perform with the execution role, see the following topics.

For more information about IAM roles, see [IAM Roles](#) in the *IAM User Guide*.

### Topics

- [CreateAutoMLJob API: Execution Role Permissions \(p. 2450\)](#)
- [CreateDomain API: Execution Role Permissions \(p. 2451\)](#)
- [CreateImage and UpdateImage APIs: Execution Role Permissions \(p. 2452\)](#)
- [CreateNotebookInstance API: Execution Role Permissions \(p. 2452\)](#)
- [CreateHyperParameterTuningJob API: Execution Role Permissions \(p. 2455\)](#)

- [CreateProcessingJob API: Execution Role Permissions \(p. 2457\)](#)
- [CreateTrainingJob API: Execution Role Permissions \(p. 2460\)](#)
- [CreateModel API: Execution Role Permissions \(p. 2462\)](#)

## CreateAutoMLJob API: Execution Role Permissions

For an execution role that you can pass in a CreateAutoMLJob API request, you can attach the following minimum permission policy to the role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole",
                "sagemaker:DescribeEndpointConfig",
                "sagemaker:DescribeModel",
                "sagemaker:InvokeEndpoint",
                "sagemaker>ListTags",
                "sagemaker:DescribeEndpoint",
                "sagemaker>CreateModel",
                "sagemaker>CreateEndpointConfig",
                "sagemaker>CreateEndpoint",
                "sagemaker>DeleteModel",
                "sagemaker>DeleteEndpointConfig",
                "sagemaker>DeleteEndpoint",
                "cloudwatch:PutMetricData",
                "logs>CreateLogStream",
                "logs:PutLogEvents",
                "logs>CreateLogGroup",
                "logs:DescribeLogStreams",
                "s3:GetObject",
                "s3:PutObject",
                "s3>ListBucket",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*"
        }
    ]
}
```

If you specify a private VPC for your AutoML job, add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
        "ec2>CreateNetworkInterface",
        "ec2>CreateNetworkInterfacePermission",
        "ec2>DeleteNetworkInterface",
        "ec2>DeleteNetworkInterfacePermission",
        "ec2:DescribeNetworkInterfacePermissions",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups"
    ]
}
```

If your input is encrypted using server-side encryption with an Amazon KMS–managed key (SSE-KMS), add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
        "kms:Decrypt"
    ]
}
```

If you specify a KMS key in the output configuration of your AutoML job, add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
        "kms:Encrypt"
    ]
}
```

If you specify a volume KMS key in the resource configuration of your AutoML job, add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
        "kms>CreateGrant"
    ]
}
```

## CreateDomain API: Execution Role Permissions

The execution role for Amazon Web Services SSO domains and the user/execution role for IAM domains need the following permissions when you pass an Amazon KMS customer managed key (CMK) as the `KmsKeyId` in the `CreateDomain` API request. The permissions are enforced during the `CreateApp` API call.

For an execution role that you can pass in the `CreateDomain` API request, you can attach the following permission policy to the role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "kms>CreateGrant",
                "kms:DescribeKey"
            ],
            "Resource": "arn:aws:kms:region:account-id:key/kms-key-id"
        },
    ]
}
```

Alternatively, if the permissions are specified in a KMS policy, you can attach the following policy to the role:

```
{
    "Sid": "Allow use of the key",
    "Effect": "Allow",
```

```

    "Principal": {
        "AWS": [
            "arn:aws:iam::account-id:role/ExecutionRole"
        ]
    },
    "Action": [
        "kms:DescribeKey",
        "kms>CreateGrant"
    ],
    "Resource": "*"
}

```

## CreatelImage and UpdatelImage APIs: Execution Role Permissions

For an execution role that you can pass in a `CreateImage` or `UpdateImage` API request, you can attach the following permission policy to the role:

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchGetImage",
                "ecr:GetDownloadUrlForLayer"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "sagemaker.amazonaws.com"
                }
            }
        }
    ]
}

```

## CreateNotebookInstance API: Execution Role Permissions

The permissions that you grant to the execution role for calling the `CreateNotebookInstance` API depend on what you plan to do with the notebook instance. If you plan to use it to invoke SageMaker APIs and pass the same role when calling the `CreateTrainingJob` and `CreateModel` APIs, attach the following permissions policy to the role:

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:*",
                "ecr:GetAuthorizationToken",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage",
                "ecr:BatchCheckLayerAvailability",
                "ecr:SetRepositoryPolicy",
                "lambda:InvokeFunction"
            ],
            "Resource": "*"
        }
    ]
}

```

```

        "ecr:CompleteLayerUpload",
        "ecr:BatchDeleteImage",
        "ecr:UploadLayerPart",
        "ecr:DeleteRepositoryPolicy",
        "ecr:InitiateLayerUpload",
        "ecr:DeleteRepository",
        "ecr:PutImage",
        "ecr>CreateRepository",
        "cloudwatch:PutMetricData",
        "cloudwatch:GetMetricData",
        "cloudwatch:GetMetricStatistics",
        "cloudwatch>ListMetrics",
        "logs>CreateLogGroup",
        "logs>CreateLogStream",
        "logs>DescribeLogStreams",
        "logs>PutLogEvents",
        "logs>GetLogEvents",
        "s3>CreateBucket",
        "s3>ListBucket",
        "s3>GetBucketLocation",
        "s3>GetObject",
        "s3>PutObject",
        "s3>DeleteObject",
        "robomaker>CreateSimulationApplication",
        "robomaker>DescribeSimulationApplication",
        "robomaker>DeleteSimulationApplication",
        "robomaker>CreateSimulationJob",
        "robomaker>DescribeSimulationJob",
        "robomaker>CancelSimulationJob",
        "ec2>CreateVpcEndpoint",
        "ec2>DescribeRouteTables",
        "fsx>DescribeFileSystem",
        "elasticfilesystem>DescribeMountTargets"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "codecommit:GitPull",
        "codecommit:GitPush"
    ],
    "Resource": [
        "arn:aws:codecommit:*::*:sagemaker*",
        "arn:aws:codecommit:*::*:SageMaker*",
        "arn:aws:codecommit:*::*:Sagemaker*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "iam:PassRole"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "iam:PassedToService": "sagemaker.amazonaws.com"
        }
    }
}
]
}

```

To tighten the permissions, limit them to specific Amazon S3 and Amazon ECR resources, by restricting `"Resource": "*"`, as follows:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:*",
                "ecr:GetAuthorizationToken",
                "cloudwatch:PutMetricData",
                "logs>CreateLogGroup",
                "logs>CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents",
                "logs:GetLogEvents"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "iam:PassedToService": "sagemaker.amazonaws.com"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3>ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket/object1",
                "arn:aws:s3:::outputbucket/path",
                "arn:aws:s3:::inputbucket/object2",
                "arn:aws:s3:::inputbucket/object3"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": [
                "arn:aws:ecr:::repository/my-repo1",
                "arn:aws:ecr:::repository/my-repo2",
                "arn:aws:ecr:::repository/my-repo3"
            ]
        }
    ]
}
```

```
    ]
}
```

If you plan to access other resources, such as Amazon DynamoDB or Amazon Relational Database Service, add the relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope the `s3>ListBucket` permission to the specific bucket that you specify as `InputDataConfig.DataSource.S3DataSource.S3Uri` in a `CreateTrainingJob` request.
- Scope `s3GetObject`, `s3PutObject`, and `s3DeleteObject` permissions as follows:
  - Scope to the following values that you specify in a `CreateTrainingJob` request:

```
InputDataConfig.DataSource.S3DataSource.S3Uri
```

```
OutputDataConfig.S3OutputPath
```

- Scope to the following values that you specify in a `CreateModel` request:

```
PrimaryContainer.ModelDataUrl
```

```
SupplementalContainers.ModelDataUrl
```

- Scope `ecr` permissions as follows:

- Scope to the `AlgorithmSpecification.TrainingImage` value that you specify in a `CreateTrainingJob` request.

- Scope to the `PrimaryContainer.Image` value that you specify in a `CreateModel` request:

The `cloudwatch` and `logs` actions are applicable for `"*"` resources. For more information, see [CloudWatch Resources and Operations](#) in the Amazon CloudWatch User Guide.

## CreateHyperParameterTuningJob API: Execution Role Permissions

For an execution role that you can pass in a `CreateHyperParameterTuningJob` API request, you can attach the following permission policy to the role:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData",
        "logs:CreateLogStream",
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:DescribeLogStreams",
        "s3:GetObject",
        "s3:PutObject",
        "s3>ListBucket",
        "ecr:GetAuthorizationToken",
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
      ],
      "Resource": "*"
    }
  ]
}
```

}

Instead of specifying "Resource": "\*", you could scope these permissions to specific Amazon S3 and Amazon ECR resources:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs>CreateLogStream",
                "logs:PutLogEvents",
                "logs>CreateLogGroup",
                "logs:DescribeLogStreams",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3>ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket/object",
                "arn:aws:s3:::outputbucket/path"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "arn:aws:ecr:::repository/my-repo"
        }
    ]
}
```

If the training container associated with the hyperparameter tuning job needs to access other data sources, such as DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope the s3>ListBucket permission to a specific bucket that you specify as the `InputDataConfig.DataSource.S3DataSource.S3Uri` in a `CreateTrainingJob` request.
- Scope the s3GetObject and s3PutObject permissions to the following objects that you specify in the input and output data configuration in a `CreateHyperParameterTuningJob` request:

`InputDataConfig.DataSource.S3DataSource.S3Uri`

`OutputDataConfig.S3OutputPath`

- Scope Amazon ECR permissions to the registry path (`AlgorithmSpecification.TrainingImage`) that you specify in a `CreateHyperParameterTuningJob` request.

The `cloudwatch` and `logs` actions are applicable for "\*" resources. For more information, see [CloudWatch Resources and Operations](#) in the Amazon CloudWatch User Guide.

If you specify a private VPC for your hyperparameter tuning job, add the following permissions:

```
{  
    "Effect": "Allow",  
    "Action": [  
        "ec2:CreateNetworkInterface",  
        "ec2:CreateNetworkInterfacePermission",  
        "ec2:DeleteNetworkInterface",  
        "ec2:DeleteNetworkInterfacePermission",  
        "ec2:DescribeNetworkInterfaces",  
        "ec2:DescribeVpcs",  
        "ec2:DescribeDhcpOptions",  
        "ec2:DescribeSubnets",  
        "ec2:DescribeSecurityGroups"  
    ]  
}
```

If your input is encrypted using server-side encryption with an Amazon KMS-managed key (SSE-KMS), add the following permissions:

```
{  
    "Effect": "Allow",  
    "Action": [  
        "kms:Decrypt"  
    ]  
}
```

If you specify a KMS key in the output configuration of your hyperparameter tuning job, add the following permissions:

```
{  
    "Effect": "Allow",  
    "Action": [  
        "kms:Encrypt"  
    ]  
}
```

If you specify a volume KMS key in the resource configuration of your hyperparameter tuning job, add the following permissions:

```
{  
    "Effect": "Allow",  
    "Action": [  
        "kms>CreateGrant"  
    ]  
}
```

## CreateProcessingJob API: Execution Role Permissions

For an execution role that you can pass in a `CreateProcessingJob` API request, you can attach the following permission policy to the role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs>CreateLogStream",
                "logs:PutLogEvents",
                "logs>CreateLogGroup",
                "logs:DescribeLogStreams",
                "s3:GetObject",
                "s3:PutObject",
                "s3>ListBucket",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*"
        }
    ]
}
```

Instead of specifying "Resource": "\*", you could scope these permissions to specific Amazon S3 and Amazon ECR resources:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs>CreateLogStream",
                "logs:PutLogEvents",
                "logs>CreateLogGroup",
                "logs:DescribeLogStreams",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3>ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket/object",
                "arn:aws:s3:::outputbucket/path"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket"
            ]
        }
    ]
}
```

```

        "Action": [
            "ecr:BatchCheckLayerAvailability",
            "ecr:GetDownloadUrlForLayer",
            "ecr:BatchGetImage"
        ],
        "Resource": "arn:aws:ecr:::repository/my-repo"
    ]
}

```

If `CreateProcessingJob.AppSpecification.ImageUri` needs to access other data sources, such as DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope the `s3>ListBucket` permission to a specific bucket that you specify as the `ProcessingInputs` in a `CreateProcessingJob` request.
- Scope the `s3GetObject` and `s3PutObject` permissions to the objects that will be downloaded or uploaded in the `ProcessingInputs` and `ProcessingOutputConfig` in a `CreateProcessingJob` request.
- Scope Amazon ECR permissions to the registry path (`AppSpecification.ImageUri`) that you specify in a `CreateProcessingJob` request.

The `cloudwatch` and `logs` actions are applicable for `"*"` resources. For more information, see [CloudWatch Resources and Operations](#) in the Amazon CloudWatch User Guide.

If you specify a private VPC for your processing job, add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:DeleteNetworkInterface",
        "ec2:DeleteNetworkInterfacePermission",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups"
    ]
}
```

If your input is encrypted using server-side encryption with an Amazon KMS–managed key (SSE-KMS), add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
        "kms:Decrypt"
    ]
}
```

If you specify a KMS key in the output configuration of your processing job, add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
        "kms:Encrypt"
    ]
}
```

```
}
```

If you specify a volume KMS key in the resource configuration of your processing job, add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
        "kms>CreateGrant"
    ]
}
```

## CreateTrainingJob API: Execution Role Permissions

For an execution role that you can pass in a `CreateTrainingJob` API request, you can attach the following permission policy to the role:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs>CreateLogStream",
                "logs:PutLogEvents",
                "logs>CreateLogGroup",
                "logs:DescribeLogStreams",
                "s3:GetObject",
                "s3:PutObject",
                "s3>ListBucket",
                "ecr:GetAuthorizationToken",
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": "*"
        }
    ]
}
```

Instead of specifying `"Resource": "*"`, you could scope these permissions to specific Amazon S3 and Amazon ECR resources:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs>CreateLogStream",
                "logs:PutLogEvents",
                "logs>CreateLogGroup",
                "logs:DescribeLogStreams",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "arn:aws:s3:::mybucket/*"
        }
    ]
}
```

```

    "Action": [
        "s3>ListBucket"
    ],
    "Resource": [
        "arn:aws:s3:::inputbucket"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:PutObject"
    ],
    "Resource": [
        "arn:aws:s3:::inputbucket/object",
        "arn:aws:s3:::outputbucket/path"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "ecr:BatchCheckLayerAvailability",
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage"
    ],
    "Resource": "arn:aws:ecr:::repository/my-repo"
}
]
}

```

If `CreateTrainingJob.AlgorithmSpecifications.TrainingImage` needs to access other data sources, such as DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope the `s3>ListBucket` permission to a specific bucket that you specify as the `InputDataConfig.DataSource.S3DataSource.S3Uri` in a `CreateTrainingJob` request.
- Scope the `s3GetObject` and `s3PutObject` permissions to the following objects that you specify in the input and output data configuration in a `CreateTrainingJob` request:

`InputDataConfig.DataSource.S3DataSource.S3Uri`

`OutputDataConfig.S3OutputPath`

- Scope Amazon ECR permissions to the registry path (`AlgorithmSpecification.TrainingImage`) that you specify in a `CreateTrainingJob` request.

The `cloudwatch` and `logs` actions are applicable for `"*"` resources. For more information, see [CloudWatch Resources and Operations](#) in the Amazon CloudWatch User Guide.

If you specify a private VPC for your training job, add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
        "ec2>CreateNetworkInterface",
        "ec2>CreateNetworkInterfacePermission",
        "ec2>DeleteNetworkInterface",
        "ec2>DeleteNetworkInterfacePermission",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeSubnets",
        ...
    ],
    "Resource": [
        ...
    ]
}
```

```
"ec2:DescribeSecurityGroups"
```

If your input is encrypted using server-side encryption with an Amazon KMS–managed key (SSE-KMS), add the following permissions:

```
{  
    "Effect": "Allow",  
    "Action": [  
        "kms:Decrypt"  
    ]  
}
```

If you specify a KMS key in the output configuration of your training job, add the following permissions:

```
{  
    "Effect": "Allow",  
    "Action": [  
        "kms:Encrypt"  
    ]  
}
```

If you specify a volume KMS key in the resource configuration of your training job, add the following permissions:

```
{  
    "Effect": "Allow",  
    "Action": [  
        "kms>CreateGrant"  
    ]  
}
```

## CreateModel API: Execution Role Permissions

For an execution role that you can pass in a `CreateModel` API request, you can attach the following permission policy to the role:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "cloudwatch:PutMetricData",  
                "logs>CreateLogStream",  
                "logs:PutLogEvents",  
                "logs>CreateLogGroup",  
                "logs:DescribeLogStreams",  
                "s3:GetObject",  
                "ecr:GetAuthorizationToken",  
                "ecr:BatchCheckLayerAvailability",  
                "ecr:GetDownloadUrlForLayer",  
                "ecr:BatchGetImage"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Instead of specifying `"Resource": "*"`, you can scope these permissions to specific Amazon S3 and Amazon ECR resources:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:PutMetricData",
                "logs>CreateLogStream",
                "logs:PutLogEvents",
                "logs>CreateLogGroup",
                "logs:DescribeLogStreams",
                "ecr:GetAuthorizationToken"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::inputbucket/object",
                "arn:aws:s3:::inputbucket/object"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "ecr:BatchCheckLayerAvailability",
                "ecr:GetDownloadUrlForLayer",
                "ecr:BatchGetImage"
            ],
            "Resource": [
                "arn:aws:ecr:::repository/my-repo",
                "arn:aws:ecr:::repository/my-repo"
            ]
        }
    ]
}
```

If `CreateModel.PrimaryContainer.Image` need to access other data sources, such as Amazon DynamoDB or Amazon RDS resources, add relevant permissions to this policy.

In the preceding policy, you scope the policy as follows:

- Scope S3 permissions to objects that you specify in the `PrimaryContainer.ModelDataUrl` in a [CreateModel](#) request.
- Scope Amazon ECR permissions to a specific registry path that you specify as the `PrimaryContainer.Image` and `SecondaryContainer.Image` in a [CreateModel](#) request.

The `cloudwatch` and `logs` actions are applicable for `"*"` resources. For more information, see [CloudWatch Resources and Operations](#) in the Amazon CloudWatch User Guide.

If you specify a private VPC for your model, add the following permissions:

```
{
    "Effect": "Allow",
    "Action": [
        "ec2>CreateNetworkInterface",
        "ec2>CreateNetworkInterfacePermission",
        "ec2>DeleteNetworkInterface",
        "ec2>DeleteNetworkInterfacePermission",
        ...
    ]
}
```

```
"ec2:DescribeNetworkInterfaces",
"ec2:DescribeVpcs",
"ec2:DescribeDhcpOptions",
"ec2:DescribeSubnets",
"ec2:DescribeSecurityGroups"
```

## Amazon Managed Policies for Amazon SageMaker

To add permissions to users, groups, and roles, it is easier to use Amazon managed policies than to write policies yourself. It takes time and expertise to [create IAM customer managed policies](#) that provide your team with only the permissions they need. To get started quickly, you can use our Amazon managed policies. These policies cover common use cases and are available in your Amazon account. For more information about Amazon managed policies, see [Amazon managed policies](#) in the *IAM User Guide*.

Amazon services maintain and update Amazon managed policies. You can't change the permissions in Amazon managed policies. Services occasionally add additional permissions to an Amazon managed policy to support new features. This type of update affects all identities (users, groups, and roles) to which the policy is attached. Services are most likely to update an Amazon managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an Amazon managed policy, so policy updates won't break your existing permissions.

Additionally, Amazon supports managed policies for job functions that span multiple services. For example, the `ReadOnlyAccess` Amazon managed policy provides read-only access to all Amazon services and resources. When a service launches a new feature, Amazon adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see [Amazon managed policies for job functions](#) in the *IAM User Guide*.

### Important

We recommend that you use the most restricted policy that allows you to perform your use case.

The following Amazon managed policies, which you can attach to users in your account, are specific to Amazon SageMaker:

- **AmazonSageMakerFullAccess** – Grants full access to Amazon SageMaker resources and the supported operations. This does not provide unrestricted Amazon S3 access, but supports buckets and objects with specific `sagemaker` tags.
- **AmazonSageMakerReadOnly** – Grants read-only access to Amazon SageMaker resources.

The following Amazon managed policies can be attached to users in your account but are not recommended:

- **AdministratorAccess** – Grants all actions for all Amazon services and for all resources in the account.
- **DataScientist** – Grants a wide range of permissions to cover most of the use cases (primarily for analytics and business intelligence) encountered by data scientists.

You can review these permissions policies by signing in to the IAM console and searching for them.

You can also create your own custom IAM policies to allow permissions for Amazon SageMaker actions and resources as you need them. You can attach these custom policies to the IAM users or groups that require them.

### Topics

- [AmazonSageMakerFullAccess \(p. 2465\)](#)
- [AmazonSageMakerReadOnly \(p. 2473\)](#)

- [SageMaker Updates to Amazon Managed Policies \(p. 2474\)](#)

## AmazonSageMakerFullAccess

This policy grants administrative permissions that allow a principal full access to all Amazon SageMaker resources and operations. The policy also provides select access to related services.

### Permissions details

This policy includes the following permissions.

- `application-autoscaling` – Allows principals to automatically scale a SageMaker real-time inference endpoint.
- `athena` – Allows principals to query a list of data catalogs, databases, and table metadata from Amazon Athena.
- `aws-marketplace` – Allows principals to view Amazon AI Marketplace subscriptions. You need this if you want to access SageMaker software subscribed in Amazon Web Services Marketplace.
- `cloudformation` – Allows principals to get Amazon CloudFormation templates for using SageMaker JumpStart solutions and Pipelines. SageMaker JumpStart creates resources necessary to run end-to-end machine learning solutions that tie SageMaker to other Amazon services. SageMaker Pipelines creates new projects that are backed by Amazon Service Catalog.
- `cloudwatch` – Allows principals to post CloudWatch metrics, interact with alarms, and upload logs to CloudWatch Logs in your account.
- `codebuild` – Allows principals to store Amazon CodeBuild artifacts for SageMaker Pipeline and Projects.
- `codemcommit` – Needed for Amazon CodeCommit integration with SageMaker notebook instances.
- `cognito-idp` – Needed for Amazon SageMaker Ground Truth to define private workforce and work teams.
- `ec2` – Needed for SageMaker to manage Amazon EC2 resources and network interfaces when you specify an Amazon VPC for your SageMaker jobs, models, endpoints, and notebook instances.
- `ecr` – Needed to pull and store Docker artifacts for Amazon SageMaker Studio (custom images), training, processing, batch inference, and inference endpoints. This is also required to use your own container in SageMaker. Additional permissions for SageMaker JumpStart solutions are required to create and remove custom images on behalf of users.
- `elastic-inference` – Allows principals to connect to Amazon Elastic Inference for using SageMaker notebook instances and endpoints.
- `elasticfilesystem` – Allows principals to access Amazon Elastic File System. This is needed for SageMaker to use data sources in Amazon Elastic File System for training machine learning models.
- `fsx` – Allows principals to access Amazon FSx. This is needed for SageMaker to use data sources in Amazon FSx for training machine learning models.
- `glue` – Needed for inference pipeline pre-processing from within SageMaker notebook instances.
- `groundtruthlabeling` – Needed for Ground Truth labeling jobs. The `groundtruthlabeling` endpoint is accessed by the Ground Truth console.
- `iam` – Needed to give the SageMaker console access to available IAM roles and create service-linked roles.
- `kms` – Needed to give the SageMaker console access to available Amazon KMS keys and retrieve them for any specified Amazon KMS aliases in jobs and endpoints.
- `lambda` – Allows principals to invoke and get a list of Amazon Lambda functions.
- `logs` – Needed to allow SageMaker jobs and endpoints to publish log streams.
- `redshift` – Allows principals to access Amazon Redshift cluster credentials.
- `redshift-data` – Allows principals to use data from Amazon Redshift to run, describe, and cancel statements; get statement results; and list schemas and tables.

- **robomaker** – Allows principals to have full access to create, get descriptions, and delete Amazon Robomaker simulation applications and jobs. This is also needed to run reinforcement learning examples on notebook instances.
- **s3** – Allows principals to have full access to Amazon S3 resources pertaining to SageMaker, but not all of Amazon S3.
- **secretsmanager** – Allows principals to have full access to Amazon Secrets Manager. The principals can securely encrypt, store, and retrieve credentials for databases and other services. This is also needed for SageMaker notebook instances with SageMaker code repositories that use GitHub.
- **servicecatalog** – Allows principals to use Amazon Service Catalog. The principals can create, get a list of, update, or terminate provisioned products, such as servers, databases, websites, or applications deployed using Amazon resources. This is needed for SageMaker JumpStart and Projects to find and read service catalog products and launch Amazon resources in user accounts.
- **sns** – Allows principals to get a list of Amazon SNS topics.
- **states** – Needed for SageMaker JumpStart and Pipelines to use a service catalog to create step function resources.
- **tag** – Needed for SageMaker Pipelines to render in Studio. Studio needs resources tagged with particular `sagemaker:project-id` tag-key. This requires the `tag:GetResources` permission.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:*"
            ],
            "NotResource": [
                "arn:aws:sagemaker:*:*:domain/*",
                "arn:aws:sagemaker:*:*:user-profile/*",
                "arn:aws:sagemaker:*:*:app/*",
                "arn:aws:sagemaker:*:*:flow-definition/*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker>CreatePresignedDomainUrl",
                "sagemaker:DescribeDomain",
                "sagemaker>ListDomains",
                "sagemaker:DescribeUserProfile",
                "sagemaker>ListUserProfiles",
                "sagemaker:App",
                "sagemaker>ListApps"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": "sagemaker:*",
            "Resource": [
                "arn:aws:sagemaker:*:*:flow-definition/*"
            ],
            "Condition": {
                "StringEqualsIfExists": {
                    "sagemaker:WorkteamType": [
                        "private-crowd",
                        "vendor-crowd"
                    ]
                }
            }
        }
    ]
}
```

```
},
{
  "Effect": "Allow",
  "Action": [
    "application-autoscaling:DeleteScalingPolicy",
    "application-autoscaling:DeleteScheduledAction",
    "application-autoscaling:DeregisterScalableTarget",
    "application-autoscaling:DescribeScalableTargets",
    "application-autoscaling:DescribeScalingActivities",
    "application-autoscaling:DescribeScalingPolicies",
    "application-autoscaling:DescribeScheduledActions",
    "application-autoscaling:PutScalingPolicy",
    "application-autoscaling:PutScheduledAction",
    "application-autoscaling:RegisterScalableTarget",
    "aws-marketplace:ViewSubscriptions",
    "cloudformation:GetTemplateSummary",
    "cloudwatch:DeleteAlarms",
    "cloudwatch:DescribeAlarms",
    "cloudwatch:GetMetricData",
    "cloudwatch:GetMetricStatistics",
    "cloudwatch>ListMetrics",
    "cloudwatch:PutMetricAlarm",
    "cloudwatch:PutMetricData",
    "codecommit:BatchGetRepositories",
    "codecommit>CreateRepository",
    "codecommit:GetRepository",
    "codecommit>List*",
    "cognito-idp:AdminAddUserToGroup",
    "cognito-idp:AdminCreateUser",
    "cognito-idp:AdminDeleteUser",
    "cognito-idp:AdminDisableUser",
    "cognito-idp:AdminEnableUser",
    "cognito-idp:AdminRemoveUserFromGroup",
    "cognito-idp>CreateGroup",
    "cognito-idp>CreateUserPool",
    "cognito-idp>CreateUserPoolClient",
    "cognito-idp>CreateUserPoolDomain",
    "cognito-idp:DescribeUserPool",
    "cognito-idp:DescribeUserPoolClient",
    "cognito-idp>List*",
    "cognito-idp:UpdateUserPool",
    "cognito-idp:UpdateUserPoolClient",
    "ec2>CreateNetworkInterface",
    "ec2>CreateNetworkInterfacePermission",
    "ec2>CreateVpcEndpoint",
    "ec2>DeleteNetworkInterface",
    "ec2>DeleteNetworkInterfacePermission",
    "ec2:DescribeDhcpOptions",
    "ec2:DescribeNetworkInterfaces",
    "ec2:DescribeRouteTables",
    "ec2:DescribeSecurityGroups",
    "ec2:DescribeSubnets",
    "ec2:DescribeVpcEndpoints",
    "ec2:DescribeVpcs",
    "ecr:BatchCheckLayerAvailability",
    "ecr:BatchGetImage",
    "ecr>CreateRepository",
    "ecr:Describe*",
    "ecr:GetAuthorizationToken",
    "ecr:GetDownloadUrlForLayer",
    "ecr:StartImageScan",
    "elastic-inference:Connect",
    "elasticfilesystem:DescribeFileSystems",
    "elasticfilesystem:DescribeMountTargets",
    "fsx:DescribeFileSystems",
    "glue>CreateJob",
```

```

        "glue>DeleteJob",
        "glue>GetJob*",
        "glue>GetTable*",
        "glue>GetWorkflowRun",
        "glue>ResetJobBookmark",
        "glue>StartJobRun",
        "glue>StartWorkflowRun",
        "glue>UpdateJob",
        "groundtruthlabeling:*",
        "iam>ListRoles",
        "kms>DescribeKey",
        "kms>ListAliases",
        "lambda>ListFunctions",
        "logs>CreateLogDelivery",
        "logs>CreateLogGroup",
        "logs>CreateLogStream",
        "logs>DeleteLogDelivery",
        "logs>Describe*",
        "logs>GetLogDelivery",
        "logs>GetLogEvents",
        "logs>ListLogDeliveries",
        "logs>PutLogEvents",
        "logs>PutResourcePolicy",
        "logs>UpdateLogDelivery",
        "robomaker>CreateSimulationApplication",
        "robomaker>DescribeSimulationApplication",
        "robomaker>DeleteSimulationApplication",
        "robomaker>CreateSimulationJob",
        "robomaker>DescribeSimulationJob",
        "robomaker>CancelSimulationJob",
        "secretsmanager>ListSecrets",
        "servicecatalog>Describe*",
        "servicecatalog>List*",
        "servicecatalog>ScanProvisionedProducts",
        "servicecatalog>SearchProducts",
        "servicecatalog>SearchProvisionedProducts",
        "sns>ListTopics",
        "tag>GetResources"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "ecr>SetRepositoryPolicy",
        "ecr>CompleteLayerUpload",
        "ecr>BatchDeleteImage",
        "ecr>UploadLayerPart",
        "ecr>DeleteRepositoryPolicy",
        "ecr>InitiateLayerUpload",
        "ecr>DeleteRepository",
        "ecr>PutImage"
    ],
    "Resource": [
        "arn:aws:ecr::*:repository/*sagemaker*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "codecommit>GitPull",
        "codecommit>GitPush"
    ],
    "Resource": [
        "arn:aws:codecommit:::sagemaker*",
        "arn:aws:codecommit:::SageMaker*",

```

```

        "arn:aws:codecommit:*::*:Sagemaker"
    ],
},
{
    "Action": [
        "codebuild:BatchGetBuilds",
        "codebuild:StartBuild"
    ],
    "Resource": [
        "arn:aws:codebuild:*:::project/sagemaker*",
        "arn:aws:codebuild:*:::build/*"
    ],
    "Effect": "Allow"
},
{
    "Action": [
        "states:DescribeExecution",
        "states:GetExecutionHistory",
        "states:StartExecution",
        "states:StopExecution",
        "states:UpdateStateMachine"
    ],
    "Resource": [
        "arn:aws:states:*:::statemachine:*sagemaker*",
        "arn:aws:states:*:::execution:*sagemaker*"
    ],
    "Effect": "Allow"
},
{
    "Effect": "Allow",
    "Action": [
        "secretsmanager:DescribeSecret",
        "secretsmanager:GetSecretValue",
        "secretsmanager>CreateSecret"
    ],
    "Resource": [
        "arn:aws:secretsmanager:*:::secret:AmazonSageMaker-*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "secretsmanager:DescribeSecret",
        "secretsmanager:GetSecretValue"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "secretsmanager:ResourceTag/SageMaker": "true"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "servicecatalog:ProvisionProduct"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "servicecatalog:TerminateProvisionedProduct",
        "servicecatalog:UpdateProvisionedProduct"
    ],
    "Resource": "*",
}

```

```

        "Condition": {
            "StringEquals": {
                "servicecatalog:userLevel": "self"
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:DeleteObject",
                "s3:AbortMultipartUpload"
            ],
            "Resource": [
                "arn:aws:s3::::*SageMaker*",
                "arn:aws:s3::::*Sagemaker*",
                "arn:aws:s3::::*sagemaker*",
                "arn:aws:s3::::aws-glue*"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "*",
            "Condition": {
                "StringEqualsIgnoreCase": {
                    "s3:ExistingObjectTag/SageMaker": "true"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/servicecatalog:provisioning": "true"
                }
            }
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3>CreateBucket",
                "s3:GetBucketLocation",
                "s3>ListBucket",
                "s3>ListAllMyBuckets",
                "s3:GetBucketCors",
                "s3:PutBucketCors"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketAcl",
                "s3:PutObjectAcl"
            ],
            "Resource": [
                "arn:aws:s3::::*SageMaker*",
                "arn:aws:s3::::*Sagemaker*",

```

```

        "arn:aws:s3::::*sagemaker*"
    ],
},
{
    "Effect": "Allow",
    "Action": [
        "lambda:InvokeFunction"
    ],
    "Resource": [
        "arn:aws:lambda::::function:*SageMaker*",
        "arn:aws:lambda::::function:*sagemaker*",
        "arn:aws:lambda::::function:*Sagemaker*",
        "arn:aws:lambda::::function:*LabelingFunction*"
    ],
},
{
    "Action": "iam:CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::::role/aws-service-role/sagemaker.application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoScaling_SageMakerEndpoint",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "sagemaker.application-autoscaling.amazonaws.com"
        }
    }
},
{
    "Effect": "Allow",
    "Action": "iam:CreateServiceLinkedRole",
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "iam:AWSServiceName": "robomaker.amazonaws.com"
        }
    }
},
{
    "Effect": "Allow",
    "Action": [
        "sns:Subscribe",
        "sns>CreateTopic"
    ],
    "Resource": [
        "arn:aws:sns::::*SageMaker*",
        "arn:aws:sns::::*Sagemaker*",
        "arn:aws:sns::::*sagemaker*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "iam:PassRole"
    ],
    "Resource": "arn:aws:iam::::role/*",
    "Condition": {
        "StringEquals": {
            "iam:PassedToService": [
                "sagemaker.amazonaws.com",
                "glue.amazonaws.com",
                "robomaker.amazonaws.com",
                "states.amazonaws.com"
            ]
        }
    }
},
{

```

```

    "Effect": "Allow",
    "Action": [
        "athena>ListDataCatalogs",
        "athena>ListDatabases",
        "athena>ListTableMetadata",
        "athena>GetQueryExecution",
        "athena>GetQueryResults",
        "athena>StartQueryExecution",
        "athena>StopQueryExecution"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "glue>CreateTable"
    ],
    "Resource": [
        "arn:aws:glue::::table/*/sagemaker_tmp_*",
        "arn:aws:glue::::table/sagemaker_featurestore/*",
        "arn:aws:glue::::catalog",
        "arn:aws:glue::::database/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "glue>DeleteTable"
    ],
    "Resource": [
        "arn:aws:glue::::table/*/sagemaker_tmp_*",
        "arn:aws:glue::::catalog",
        "arn:aws:glue::::database/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "glue>GetDatabases",
        "glue>GetTable",
        "glue>GetTables"
    ],
    "Resource": [
        "arn:aws:glue::::table/*",
        "arn:aws:glue::::catalog",
        "arn:aws:glue::::database/*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "glue>CreateDatabase",
        "glue>GetDatabase"
    ],
    "Resource": [
        "arn:aws:glue::::catalog",
        "arn:aws:glue::::database/sagemaker_featurestore",
        "arn:aws:glue::::database/sagemaker_processing",
        "arn:aws:glue::::database/default",
        "arn:aws:glue::::database/sagemaker_data_wrangler"
    ]
},
{
    "Effect": "Allow",

```

```

    "Action": [
        "redshift-data:ExecuteStatement",
        "redshift-data:DescribeStatement",
        "redshift-data:CancelStatement",
        "redshift-data:GetStatementResult",
        "redshift-data>ListSchemas",
        "redshift-data>ListTables"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Effect": "Allow",
    "Action": [
        "redshift:GetClusterCredentials"
    ],
    "Resource": [
        "arn:aws:redshift:*:dbuser:*:sagemaker_access*",
        "arn:aws:redshift:*:dbname:*
    ]
}
]
}

```

## AmazonSageMakerReadOnly

This policy provides read-only access to SageMaker via the Amazon Web Services Management Console and SDK.

### Permissions details

This policy includes the following permissions.

- `application-autoscaling` – Allows users to browse descriptions of scalable SageMaker real-time inference endpoints.
- `aws-marketplace` – Allows users to view Amazon AI Marketplace subscriptions.
- `cloudwatch` – Allows users to receive CloudWatch alarms.
- `cognito-idp` – Needed for Amazon SageMaker Ground Truth to browse descriptions and lists of private workforce and work teams.
- `ecr` – Needed to read Docker artifacts for training and inference.

```

{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sagemaker:Describe*",
                "sagemaker>List*",
                "sagemaker:BatchGetMetrics",
                "sagemaker:BatchGetRecord",
                "sagemaker:GetDeviceRegistration",
                "sagemaker:GetDeviceFleetReport",
                "sagemaker:GetSearchSuggestions",
                "sagemaker:GetRecord",
                "sagemaker:Search"
            ],
            "Resource": "*"
        }
    ]
}

```

```

        },
        {
            "Effect": "Allow",
            "Action": [
                "application-autoscaling:DescribeScalableTargets",
                "application-autoscaling:DescribeScalingActivities",
                "application-autoscaling:DescribeScalingPolicies",
                "application-autoscaling:DescribeScheduledActions",
                "aws-marketplace:ViewSubscriptions",
                "cloudwatch:DescribeAlarms",
                "cognito-idp:DescribeUserPool",
                "cognito-idp:DescribeUserPoolClient",
                "cognito-idp>ListGroups",
                "cognito-idp>ListIdentityProviders",
                "cognito-idp>ListUserPoolClients",
                "cognito-idp>ListUserPools",
                "cognito-idp>ListUsers",
                "cognito-idp>ListUsersInGroup",
                "ecr:Describe*"
            ],
            "Resource": "*"
        }
    ]
}

```

## SageMaker Updates to Amazon Managed Policies

View details about updates to Amazon managed policies for SageMaker since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the [SageMaker document history](#) page.

Change	Description	Date
AmazonSageMakerReadOnly updates – Update to an existing policy	New API <code>BatchGetRecord</code> added for SageMaker Feature Store.	JUNE 10, 2021
SageMaker started tracking changes	SageMaker started tracking changes for its Amazon managed policies.	JUNE 01, 2021

## Amazon SageMaker API Permissions: Actions, Permissions, and Resources Reference

When you are setting up access control and writing a permissions policy that you can attach to an IAM identity (an identity-based policy), use the following as a reference. For each Amazon SageMaker API operation, the corresponding actions for which you can grant permissions to perform the action, and the Amazon resource for which you can grant the permissions. You specify the actions in the policy's `Action` field, and you specify the resource value in the policy's `Resource` field.

**Note**

Except for the `ListTags` API, resource-level restrictions are not available on `List-` calls. Any user calling a `List-` API will see all resources of that type in the account.

To express conditions in your Amazon SageMaker policies, you can use Amazon-wide condition keys. For a complete list of Amazon-wide keys, see [Available Keys](#) in the *IAM User Guide*.

## Amazon SageMaker API Operations and Required Permissions for Actions

Amazon SageMaker API Operations	Required Permissions (API Actions)	Resources
AddTags	sagemaker:AddTags	arn:aws:sagemaker: <i>region:account-id</i> :*
CreateAutoMLJob	sagemaker:CreateAutoMLJob iam:PassRole  The following permission is required only if the associated ResourceConfig has a specified VolumeKmsKeyId and the associated role does not have a policy that permits this action:  kms:CreateGrant	arn:aws:sagemaker: <i>region:account-id:automl-job/autoMLJobName</i>
CreateDomain	sagemaker:CreateDomain iam:CreateServiceLinkedRole iam:PassRole  Required if a KMS customer managed CMK is specified for KmsKeyId:  elasticfilesystem>CreateFileSystem  kms:CreateGrant  kms:Decrypt  kms:DescribeKey  kms:GenerateDataKeyWithoutPlainText	arn:aws:sagemaker: <i>region:account-id:domain/domain-id</i>
CreateEndpoint	sagemaker:CreateEndpoint  kms:CreateGrant (required only if the associated EndPointConfig has a KmsKeyId specified)	arn:aws:sagemaker: <i>region:account-id:endpoint/endpointName</i>  arn:aws:sagemaker: <i>region:account-id:endpoint-config/endpointConfigName</i>
CreateEndpointConfig	sagemaker:CreateEndpointConfig	arn:aws:sagemaker: <i>region:account-id:endpoint-config/endpointConfigName</i>
CreateFlowDefinition	sagemaker:CreateFlowDefinition  iam:PassRole	arn:aws:sagemaker: <i>region:account-id:flow-definition/flowDefinitionName</i>
CreateHumanTaskUi	sagemaker:CreateHumanTaskUi	arn:aws:sagemaker: <i>region:account-id:human-task-ui/humanTaskUiName</i>

Amazon SageMaker API Operations	Required Permissions (API Actions)	Resources
CreateHyperParameter	sagemaker:CreateHyperParameter iam:PassRole  The following permission is required only if any of the associated ResourceConfig have a specified VolumeKmsKeyId and the associated role does not have a policy that permits this action:  kms:CreateGrant	arn:aws:sagemaker:region:account-id:hyper-parameter-tuning-job/ <i>hyperParameterTuningJobName</i>
CreateImage	sagemaker:CreateImage iam:PassRole	arn:aws:sagemaker:region:account-id:image/*
CreateImageVersion	sagemaker:CreateImageVersion	arn:aws:sagemaker:region:account-id:image-version/ <i>imageName</i> /*
CreateLabelingJob	sagemaker:CreateLabelingJob iam:PassRole	arn:aws:sagemaker:region:account-id:labeling-job/ <i>labelingJobName</i>
CreateModel	sagemaker:CreateModel iam:PassRole	arn:aws:sagemaker:region:account-id:model/ <i>modelName</i>

Amazon SageMaker API Operations	Required Permissions (API Actions)	Resources
<a href="#">CreateNotebookInstance</a>	<p>sagemaker:CreateNotebookInstance iam:PassRole</p> <p>The following permissions are required only if you specify a VPC for your notebook instance:</p> <ul style="list-style-type: none"> <li>ec2:CreateNetworkInterface</li> <li>ec2:DescribeSecurityGroups</li> <li>ec2:DescribeSubnets</li> <li>ec2:DescribeVpcs</li> </ul> <p>The following permission is required only if you specify a VPC and an elastic inference accelerator for your notebook instance:</p> <ul style="list-style-type: none"> <li>ec2:DescribeVpcEndpoints</li> </ul> <p>The following permissions are required only if you specify an encryption key:</p> <ul style="list-style-type: none"> <li>kms:DescribeKey</li> <li>kms&gt;CreateGrant</li> </ul> <p>The following permission is required only if you specify an Amazon Secrets Manager secret to access a private Git repository:</p> <ul style="list-style-type: none"> <li>secretsmanager:GetSecretValue</li> </ul>	arn:aws:sagemaker: <i>region:account-id:notebook-instance/notebookInstanceName</i>
<a href="#">CreatePresignedNotebookInstanceUrl</a>	sagemaker:CreatePresignedNotebookInstanceUrl	arn:aws:sagemaker: <i>region:account-id:notebook-instance/notebookInstanceName</i>
<a href="#">CreateProcessingJob</a>	<p>sagemaker:CreateProcessingJob iam:PassRole</p> <p>kms:CreateGrant (required only if the associated ProcessingResources has a specified VolumeKmsKeyId and the associated role does not have a policy that permits this action)</p>	arn:aws:sagemaker: <i>region:account-id:processing-job/processingJobName</i>

Amazon SageMaker API Operations	Required Permissions (API Actions)	Resources
<a href="#">CreateTrainingJob</a>	sagemaker:CreateTrainingJob  iam:PassRole  kms:CreateGrant (required only if the associated ResourceConfig has a specified VolumeKmsKeyId and the associated role does not have a policy that permits this action)	arn:aws:sagemaker: <i>region:account-id:training-job/trainingJobName</i>
<a href="#">CreateTransformJob</a>	sagemaker:CreateTransformJob  kms:CreateGrant (required only if the associated TransformResources has a specified VolumeKmsKeyId and the associated role does not have a policy that permits this action)	arn:aws:sagemaker: <i>region:account-id:transform-job/transformJobName</i>
<a href="#">CreateWorkforce</a>	sagemaker>CreateWorkforce  cognito-idp:DescribeUserPoolClient  cognito-idp:UpdateUserPool  cognito-idp:DescribeUserPool  cognito-idp:UpdateUserPoolClient	arn:aws:sagemaker: <i>region:account-id:workforce/*</i>
<a href="#">CreateWorkteam</a>	sagemaker>CreateWorkteam  cognito-idp:DescribeUserPoolClient  cognito-idp:UpdateUserPool  cognito-idp:DescribeUserPool  cognito-idp:UpdateUserPoolClient	arn:aws:sagemaker: <i>region:account-id:workteam/private-crowd/work team name</i>
<a href="#">DeleteEndpoint</a>	sagemaker>DeleteEndpoint	arn:aws:sagemaker: <i>region:account-id:endpoint/endpointName</i>
<a href="#">DeleteEndpointConfig</a>	sagemaker>DeleteEndpointConfig	arn:aws:sagemaker: <i>region:account-id:endpoint-config/endpointConfigName</i>
<a href="#">DeleteFlowDefinition</a>	sagemaker>DeleteFlowDefinition	arn:aws:sagemaker: <i>region:account-id:flow-definition/flowDefinitionName</i>

Amazon SageMaker API Operations	Required Permissions (API Actions)	Resources
<a href="#">DeleteHumanLoop</a>	sagemaker:DeleteHumanLoop	arn:aws:sagemaker: <i>region:account-id:human-loop/humanLoopName</i>
<a href="#">DeleteImage</a>	sagemaker:DeleteImage	arn:aws:sagemaker: <i>region:account-id:image/imageName</i>
<a href="#">DeleteImageVersion</a>	sagemaker:DeleteImageVersion	arn:aws:sagemaker: <i>region:account-id:image-version/imageName/versionNumber</i>
<a href="#">DeleteModel</a>	sagemaker:DeleteModel	arn:aws:sagemaker: <i>region:account-id:model/modelName</i>
<a href="#">DeleteNotebookInstance</a>	sagemaker:DeleteNotebookInstance  The following permission is required only if you specified a VPC for your notebook instance:  ec2:DeleteNetworkInterface  The following permissions are required only if you specified an encryption key when you created the notebook instance:  kms:DescribeKey	arn:aws:sagemaker: <i>region:account-id:notebook-instance/notebookInstanceName</i>
<a href="#">DeleteTags</a>	sagemaker:DeleteTags	arn:aws:sagemaker: <i>region:account-id:*</i>
<a href="#">DeleteWorkforce</a>	sagemaker:DeleteWorkforce	arn:aws:sagemaker: <i>region:account-id:workforce/*</i>
<a href="#">DeleteWorkteam</a>	sagemaker:DeleteWorkteam	arn:aws:sagemaker: <i>region:account-id:workteam/private-crowd/*</i>
<a href="#">DescribeEndpoint</a>	sagemaker:DescribeEndpoint	arn:aws:sagemaker: <i>region:account-id:endpoint/endpointName</i>
<a href="#">DescribeEndpointConfig</a>	sagemaker:DescribeEndpointConfig	arn:aws:sagemaker: <i>region:account-id:endpoint-config/endpointConfigName</i>
<a href="#">DescribeFlowDefinition</a>	sagemaker:DescribeFlowDefinition	arn:aws:sagemaker: <i>region:account-id:flow-definition/flowDefinitionName</i>
<a href="#">DescribeHumanLoop</a>	sagemaker:DescribeHumanLoop	arn:aws:sagemaker: <i>region:account-id:human-loop/humanLoopName</i>
<a href="#">DescribeHumanTaskUi</a>	sagemaker:DescribeHumanTaskUi	arn:aws:sagemaker: <i>region:account-id:human-task-ui/humanTaskUiName</i>

Amazon SageMaker API Operations	Required Permissions (API Actions)	Resources
DescribeHyperParameters	sagemaker:DescribeHyperParameters	arn:aws:sagemaker: <b>region:account-id:hyper-parameter-tuning-job/<i>hyperParameterTuningJob</i></b>
DescribeImage	sagemaker:DescribeImage	arn:aws:sagemaker: <b>region:account-id:image/<i>imageName</i></b>
DescribeImageVersions	sagemaker:DescribeImageVersions	arn:aws:sagemaker: <b>region:account-id:image-version/<i>imageName</i>/<i>versionNumber</i></b>
DescribeLabelingJob	sagemaker:DescribeLabelingJob	arn:aws:sagemaker: <b>region:account-id:labeling-job/<i>labelingJobName</i></b>
DescribeModel	sagemaker:DescribeModel	arn:aws:sagemaker: <b>region:account-id:model/<i>modelName</i></b>
DescribeNotebookInstance	sagemaker:DescribeNotebookInstance	arn:aws:sagemaker: <b>region:account-id:notebook-instance/<i>notebookInstanceName</i></b>
DescribeProcessingJob	sagemaker:DescribeProcessingJob	arn:aws:sagemaker: <b>region:account-id:processing-job/<i>processingjobname</i></b>
DescribeSubscribedWorkteams	sagemaker:DescribeSubscribedWorkteams	arn:aws:sagemaker: <b>region:account-id:workteam/vendor-crowd/*</b> aws-marketplace:ViewSubscriptions
DescribeTrainingJob	sagemaker:DescribeTrainingJob	arn:aws:sagemaker: <b>region:account-id:training-job/<i>trainingjobname</i></b>
DescribeTransformJob	sagemaker:DescribeTransformJob	arn:aws:sagemaker: <b>region:account-id:transform-job/<i>transformjobname</i></b>
DescribeWorkforce	sagemaker:DescribeWorkforce	arn:aws:sagemaker: <b>region:account-id:workforce/*</b>
DescribeWorkteam	sagemaker:DescribeWorkteam	arn:aws:sagemaker: <b>region:account-id:workteam/private-crowd/*</b>
runtime_InvokeEndpoint	sagemaker:InvokeEndpoint	arn:aws:sagemaker: <b>region:account-id:endpoint/<i>endpointName</i></b>
ListEndpointConfigs	sagemaker>ListEndpointConfigs*	
ListEndpoints	sagemaker>ListEndpoints	*
ListFlowDefinitions	sagemaker>ListFlowDefinitions*	
ListHumanLoops	sagemaker>ListHumanLoops	*

Amazon SageMaker API Operations	Required Permissions (API Actions)	Resources
<a href="#">ListHumanTaskUis</a>	sagemaker>ListHumanTaskUis	*
<a href="#">ListHyperParameterTuningJobs</a>	sagemaker>ListHyperParameterTuningJobs	sagemaker: <code>region:account-id:hyper-parameter-tuning-job/hyperParameterTuningJob</code>
<a href="#">ListImages</a>	sagemaker>ListImages	*
<a href="#">ListImageVersions</a>	sagemaker>ListImageVersions	arn:aws:sagemaker: <code>region:account-id:image/*</code>
<a href="#">ListLabelingJobs</a>	sagemaker>ListLabelingJobs	*
<a href="#">ListLabelingJobsForWorkteam</a>	sagemaker>ListLabelingJobForWorkteam	
<a href="#">ListModels</a>	sagemaker>ListModels	*
<a href="#">ListNotebookInstances</a>	sagemaker>ListNotebookInstances	
<a href="#">ListProcessingJobs</a>	sagemaker>ListProcessingJobs	*
<a href="#">ListSubscribedWorkteams</a>	sagemaker>ListSubscribedWorkteams  aws-marketplace:ViewSubscriptions	
<a href="#">ListTags</a>	sagemaker>ListTags	arn:aws:sagemaker: <code>region:account-id:*</code>
<a href="#">ListTrainingJobs</a>	sagemaker>ListTrainingJobs	*
<a href="#">ListTrainingJobsForHyperParameterTuningJob</a>	sagemaker>ListTrainingJobsForHyperParameterTuningJob	<code>HyperParameterTuningJob:account-id:hyper-parameter-tuning-job/hyperParameterTuningJob</code>
<a href="#">ListTransformJobs</a>	sagemaker>ListTransformJobs	*
<a href="#">ListWorkforces</a>	sagemaker>ListWorkforces	*
<a href="#">ListWorkteams</a>	sagemaker>ListWorkteams	*
<a href="#">StartHumanLoop</a>	sagemaker>StartHumanLoop	arn:aws:sagemaker: <code>region:account-id:human-loop/humanLoopName</code>

Amazon SageMaker API Operations	Required Permissions (API Actions)	Resources
<a href="#">StartNotebookInstance</a>	<p>sagemaker:StartNotebookInstance iam:PassRole</p> <p>The following permissions are required only if you specified a VPC when you created your notebook instance:</p> <ul style="list-style-type: none"> <li>ec2:CreateNetworkInterface</li> <li>ec2:DescribeNetworkInterfaces</li> <li>ec2:DescribeSecurityGroups</li> <li>ec2:DescribeSubnets</li> <li>ec2:DescribeVpcs</li> </ul> <p>The following permission is required only if you specify a VPC and an elastic inference accelerator for your notebook instance:</p> <ul style="list-style-type: none"> <li>ec2:DescribeVpcEndpoints</li> </ul> <p>The following permissions are required only if you specified an encryption key when you created the notebook instance:</p> <ul style="list-style-type: none"> <li>kms:DescribeKey</li> <li>kms&gt;CreateGrant</li> </ul> <p>The following permission is required only if you specified an Amazon Secrets Manager secret to access a private Git repository when you created the notebook instance:</p> <ul style="list-style-type: none"> <li>secretsmanager:GetSecretValue</li> </ul>	arn:aws:sagemaker: <i>region:account-id:notebook-instance/notebookInstanceName</i>
<a href="#">StopHumanLoop</a>	sagemaker:StopHumanLoop	arn:aws:sagemaker: <i>region:account-id:human-loop/humanLoopName</i>
<a href="#">StopHyperParameterTuningJob</a>	sagemaker:StopHyperParameterTuningJob	arn:aws:sagemaker: <i>region:account-id:hyper-parameter-tuning-job/hyperParameterTuningJob</i>
<a href="#">StopLabelingJob</a>	sagemaker:StopLabelingJob	arn:aws:sagemaker: <i>region:account-id:labeling-job/labelingJobName</i>
<a href="#">StopNotebookInstance</a>	sagemaker:StopNotebookInstance	arn:aws:sagemaker: <i>region:account-id:notebook-instance/notebookInstanceName</i>

Amazon SageMaker API Operations	Required Permissions (API Actions)	Resources
StopProcessingJob	sagemaker:StopProcessingJob	arn:aws:sagemaker: <i>region:account-id:processing-job/processingJobName</i>
StopTrainingJob	sagemaker:StopTrainingJob	arn:aws:sagemaker: <i>region:account-id:training-job/trainingJobName</i>
StopTransformJob	sagemaker:StopTransformJob	arn:aws:sagemaker: <i>region:account-id:transform-job/transformJobName</i>
UpdateEndpoint	sagemaker:UpdateEndpoint	arn:aws:sagemaker: <i>region:account-id:endpoint/endpointName</i>
UpdateEndpointWeight	sagemaker:UpdateEndpointWeight	arn:aws:sagemaker: <i>region:account-id:endpoint/endpointName</i>
UpdateImage	sagemaker:UpdateImage iam:PassRole	arn:aws:sagemaker: <i>region:account-id:image/imageName</i>
UpdateNotebookInstance	sagemaker:UpdateNotebookInstance iam:PassRole	arn:aws:sagemaker: <i>region:account-id:notebook-instance/notebookInstanceName</i>
UpdateWorkforce	sagemaker:UpdateWorkforce	arn:aws:sagemaker: <i>region:account-id:workforce/*</i>
UpdateWorkteam	sagemaker:UpdateWorkteam	arn:aws:sagemaker: <i>region:account-id:workteam/private-crowd/*</i>

## Amazon SageMaker API and Required Permissions for Actions

### API Operation: [AddTags](#)

Required Permissions (API Action): sagemaker:AddTags

Resources: \*

### API Operation: [CreateEndpoint](#)

Required Permissions (API Action): sagemaker>CreateEndpoint

Resources: arn:aws:sagemaker:*region:account-id:endpoint/endpointName*

### API Operation: [CreateEndpointConfig](#)

Required Permissions (API Action): sagemaker>CreateEndpointConfig

Resources: arn:aws:sagemaker:*region:account-id:endpoint-config/endpointConfigName*

### API Operation: [CreateModel](#)

Required Permissions (API Action): sagemaker>CreateModel, iam:PassRole

Resources: arn:aws:sagemaker:*region:account-id:model/modelName*

**API Operation:** [CreateLabelingJob](#)

Required Permissions (API Action): `sagemaker:CreateLabelingJob, iam:PassRole`

Resources: `arn:aws:sagemaker:region:account-id:labeling-job/labelingJobName`

**API Operation:** [CreateNotebookInstance](#)

Required Permissions (API Action): `sagemaker>CreateNotebookInstance, iam:PassRole, ec2:CreateNetworkInterface, ec2:AttachNetworkInterface, ec2:ModifyNetworkInterfaceAttribute, ec2:DescribeAvailabilityZones, ec2:DescribeInternetGateways, ec2:DescribeSecurityGroups, ec2:DescribeSubnets, ec2:DescribeVpcs, kms>CreateGrant`

Resources: `arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName`

**API Operation:** [CreateTrainingJob](#)

Required Permissions (API Action): `sagemaker:CreateTrainingJob, iam:PassRole`

Resources: `arn:aws:sagemaker:region:account-id:training-job/trainingJobName`

**API Operation:** [CreateWorkforce](#)

Required Permissions (API Action): `sagemaker>CreateWorkforce, cognito-idp:DescribeUserPoolClient, cognito-idp:UpdateUserPool, cognito-idp:DescribeUserPool, cognito-idp:UpdateUserPoolClient`

Resources: `arn:aws:sagemaker:region:account-id:workforce/*`

**API Operation:** [CreateWorkteam](#)

Required Permissions (API Action): `sagemaker>CreateWorkteam, cognito-idp:DescribeUserPoolClient, cognito-idp:UpdateUserPool, cognito-idp:DescribeUserPool, cognito-idp:UpdateUserPoolClient`

Resources: `arn:aws:sagemaker:region:account-id:workteam/private-crowd/work team name`

**API Operation:** [DeleteEndpoint](#)

Required Permissions (API Action): `sagemaker>DeleteEndpoint`

Resources: `arn:aws:sagemaker:region:account-id:endpoint/endpointName`

**API Operation:** [DeleteEndpointConfig](#)

Required Permissions (API Action): `sagemaker>DeleteEndpointConfig`

Resources: `arn:aws:sagemaker:region:account-id:endpoint-config/endpointConfigName`

**API Operation:** [DeleteModel](#)

Required Permissions (API Action): `sagemaker>DeleteModel`

Resources: `arn:aws:sagemaker:region:account-id:model/modelName`

**API Operation:** [DeleteNotebookInstance](#)

Required Permissions (API Action): `sagemaker>DeleteNotebookInstance, ec2>DeleteNetworkInterface, ec2:DetachNetworkInterface, ec2:DescribeAvailabilityZones, ec2:DescribeInternetGateways, ec2:DescribeSecurityGroups, ec2:DescribeSubnets, ec2:DescribeVpcs`

Resources: `arn:aws:sagemaker:region:account-id:notebook-instance/notebookInstanceName`

**API Operation:** [DeleteTags](#)

Required Permissions (API Action): sagemaker:DeleteTags

Resources: \*

**API Operation:** [DeleteWorkteam](#)

Required Permissions (API Action): sagemaker:DeleteWorkforce

Resources: arn:aws:sagemaker:*region:account-id:workforce/private-crowd/\**

**API Operation:** [DeleteWorkteam](#)

Required Permissions (API Action): sagemaker:DeleteWorkteam

Resources: arn:aws:sagemaker:*region:account-id:workteam/private-crowd/\**

**API Operation:** [DescribeEndpoint](#)

Required Permissions (API Action): sagemaker:DescribeEndpoint

Resources: arn:aws:sagemaker:*region:account-id:endpoint/endpointName*

**API Operation:** [DescribeEndpointConfig](#)

Required Permissions (API Action): sagemaker:DescribeEndpointConfig

Resources: arn:aws:sagemaker:*region:account-id:endpoint-config/endpointConfigName*

**API Operation:** [DescribeLabelingJob](#)

Required Permissions (API Action): sagemaker:DescribeLabelingJob

Resources: arn:aws:sagemaker:*region:account-id:labeling-job/labelingJobName*

**API Operation:** [DescribeModel](#)

Required Permissions (API Action): sagemaker:DescribeModel

Resources: arn:aws:sagemaker:*region:account-id:model/modelName*

**API Operation:** [DescribeNotebookInstance](#)

Required Permissions (API Action): sagemaker:DescribeNotebookInstance

Resources: arn:aws:sagemaker:*region:account-id:notebook-instance/notebookInstanceName*

**API Operation:** [DescribeSubscribedWorkforce](#)

Required Permissions (API Action): sagemaker:DescribeSubscribedWorkforce, aws-marketplace:ViewSubscriptions

Resources: arn:aws:sagemaker:*region:account-id:workforce/\**

**API Operation:** [DescribeSubscribedWorkteam](#)

Required Permissions (API Action): sagemaker:DescribeSubscribedWorkteam, aws-marketplace:ViewSubscriptions

Resources: arn:aws:sagemaker:*region:account-id:workteam/vendor-crowd/\**

**API Operation:** [DescribeTrainingJob](#)

Required Permissions (API Action): sagemaker:DescribeTrainingJob

Resources: arn:aws:sagemaker:*region:account-id:training-job/trainingJobName*

**API Operation:** [DescribeWorkteam](#)

Required Permissions (API Action): sagemaker:DescribeWorkteam

Resources: arn:aws:sagemaker:*region:account-id*:workteam/private-crowd/\*

**API Operation:** [CreatePresignedNotebookInstanceUrl](#)

Required Permissions (API Action): sagemaker>CreatePresignedNotebookInstanceUrl

Resources: arn:aws:sagemaker:*region:account-id*:notebook-instance/*notebookInstanceName*

**API Operation:** [runtime\\_InvokeEndpoint](#)

Required Permissions (API Action): sagemaker:InvokeEndpoint

Resources: arn:aws:sagemaker:*region:account-id*:endpoint/*endpointName*

**API Operation:** [ListEndpointConfigs](#)

Required Permissions (API Action): sagemaker>ListEndpointConfigs

Resources: \*

**API Operation:** [ListEndpoints](#)

Required Permissions (API Action): sagemaker>ListEndpoints

Resources: \*

**API Operation:** [ListLabelingJobs](#)

Required Permissions (API Action): sagemaker>ListLabelingJobs

Resources: \*

**API Operation:** [ListLabelingJobsForWorkteam](#)

Required Permissions (API Action): sagemaker>ListLabelingJobsForWorkteam

Resources: \*

**API Operation:** [ListModels](#)

Required Permissions (API Action): sagemaker>ListModels

Resources: \*

**API Operation:** [ListNotebookInstances](#)

Required Permissions (API Action): sagemaker>ListNotebookInstances

Resources: \*

**API Operation:** [ListSubscribedWorkteams](#)

Required Permissions (API Action): sagemaker>ListSubscribedWorkteam, aws-marketplace:ViewSubscriptions

Resources: \*

**API Operation:** [ListTags](#)

Required Permissions (API Action): sagemaker>ListTags

Resources: \*

**API Operation:** [ListTrainingJobs](#)

Required Permissions (API Action): sagemaker>ListTrainingJobs

Resources: \*

**API Operation:** [ListWorkteams](#)

Required Permissions (API Action): sagemaker:ListWorkforces

Resources: \*

**API Operation:** [ListWorkteams](#)

Required Permissions (API Action): sagemaker>ListWorkteams

Resources: \*

**API Operation:** [StartNotebookInstance](#)

Required Permissions (API Action): sagemaker:StartNotebookInstance, iam:PassRole, ec2>CreateNetworkInterface, ec2:AttachNetworkInterface, ec2:ModifyNetworkInterfaceAttribute, ec2:DescribeAvailabilityZones, ec2:DescribeInternetGateways, ec2:DescribeSecurityGroups, ec2:DescribeSubnets, ec2:DescribeVpcs, kms>CreateGrant

Resources: arn:aws:sagemaker:*region:account-id*:notebook-instance/*notebookInstanceName*

**API Operation:** [StopLabelingJob](#)

Required Permissions (API Action): sagemaker:StopLabelingJob

Resources: arn:aws:sagemaker:*region:account-id*:labeling-job/*labelingJobName*

**API Operation:** [StopNotebookInstance](#)

Required Permissions (API Action): sagemaker:StopNotebookInstance

Resources: arn:aws:sagemaker:*region:account-id*:notebook-instance/*notebookInstanceName*

**API Operation:** [StopTrainingJob](#)

Required Permissions (API Action): sagemaker:StopTrainingJob

Resources: arn:aws:sagemaker:*region:account-id*:training-job/*trainingJobName*

**API Operation:** [UpdateEndpoint](#)

Required Permissions (API Action): sagemaker:UpdateEndpoints

Resources: arn:aws:sagemaker:*region:account-id*:endpoint/*endpointName*

**API Operation:** [UpdateNotebookInstance](#)

Required Permissions (API Action): sagemaker:UpdateNotebookInstance, iam:PassRole

Resources: arn:aws:sagemaker:*region:account-id*:notebook-instance/*notebookInstanceName*

**API Operation:** [UpdateWorkteam](#)

Required Permissions (API Action): sagemaker:UpdateWorkteam

Resources: arn:aws:sagemaker:*region:account-id*:workteam/private-crowd/\*

## Troubleshooting Amazon SageMaker Identity and Access

Use the following information to help you diagnose and fix common issues that you might encounter when working with SageMaker and IAM.

## Topics

- [I Am Not Authorized to Perform an Action in SageMaker \(p. 2488\)](#)
- [I Am Not Authorized to Perform iam:PassRole \(p. 2488\)](#)
- [I Want to View My Access Keys \(p. 2488\)](#)
- [I'm an Administrator and Want to Allow Others to Access SageMaker \(p. 2489\)](#)
- [I Want to Allow People Outside of My Amazon Account to Access My SageMaker Resources \(p. 2489\)](#)

## I Am Not Authorized to Perform an Action in SageMaker

If the Amazon Web Services Management Console tells you that you're not authorized to perform an action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a training job but does not have `sagemaker:DescribeTrainingJob` permissions.

```
User: arn:aws-cn:iam::123456789012:user/mateojackson is not
      authorized to perform: sagemaker:DescribeTrainingJob on resource: my-example-
      widget
```

In this case, Mateo asks his administrator to update his policies to allow him to access the `TrainingJob` resource using the `sagemaker:DescribeTrainingJob` action.

## I Am Not Authorized to Perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, then you must contact your administrator for assistance. Your administrator is the person that provided you with your user name and password. Ask that person to update your policies to allow you to pass a role to SageMaker.

Some Amazon services allow you to pass an existing role to that service, instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in SageMaker. However, the action requires the service to have permissions granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws-cn:iam::123456789012:user/marymajor is not authorized to perform:
      iam:PassRole
```

In this case, Mary asks her administrator to update her policies to allow her to perform the `iam:PassRole` action.

## I Want to View My Access Keys

After you create your IAM user access keys, you can view your access key ID at any time. However, you can't view your secret access key again. If you lose your secret key, you must create a new access key pair.

Access keys consist of two parts: an access key ID (for example, `AKIAIOSFODNN7EXAMPLE`) and a secret access key (for example, `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`). Like a user name and password, you must use both the access key ID and secret access key together to authenticate your requests. Manage your access keys as securely as you do your user name and password.

**Important**

Do not provide your access keys to a third party, even to help [find your canonical user ID](#). By doing this, you might give someone permanent access to your account.

When you create an access key pair, you are prompted to save the access key ID and secret access key in a secure location. The secret access key is available only at the time you create it. If you lose your secret access key, you must add new access keys to your IAM user. You can have a maximum of two access keys. If you already have two, you must delete one key pair before creating a new one. To view instructions, see [Managing access keys](#) in the *IAM User Guide*.

## I'm an Administrator and Want to Allow Others to Access SageMaker

To allow others to access SageMaker, you must create an IAM entity (user or role) for the person or application that needs access. They will use the credentials for that entity to access Amazon. You must then attach a policy to the entity that grants them the correct permissions in SageMaker.

To get started right away, see [Creating your first IAM delegated user and group](#) in the *IAM User Guide*.

## I Want to Allow People Outside of My Amazon Account to Access My SageMaker Resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether SageMaker supports these features, see [How Amazon SageMaker Works with IAM \(p. 2421\)](#).
- To learn how to provide access to your resources across Amazon accounts that you own, see [Providing access to an IAM user in another Amazon account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party Amazon accounts, see [Providing access to Amazon accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see [How IAM roles differ from resource-based policies](#) in the *IAM User Guide*.

## Logging and Monitoring

You can monitor Amazon SageMaker using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds and send notifications or take actions when those thresholds are met. For more information, see [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 2523\)](#).

Amazon CloudWatch Logs enables you to monitor, store, and access your log files from Amazon EC2 instances, Amazon CloudTrail, and other sources. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold

that you specify. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 2534\)](#).

Amazon CloudTrail provides a record of actions taken by a user, role, or an Amazon service in SageMaker. Using the information collected by CloudTrail, you can determine the request that was made to SageMaker, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, [Log Amazon SageMaker API Calls with Amazon CloudTrail \(p. 2535\)](#).

**Note**

CloudTrail does not monitor calls to [`runtime\_InvokeEndpoint`](#).

You can create rules in Amazon CloudWatch Events to react to status changes in status in a SageMaker training, hyperparameter tuning, or batch transform job. For more information, see [Automating Amazon SageMaker with Amazon EventBridge \(p. 2537\)](#).

## Compliance Validation for Amazon SageMaker

Third-party auditors assess the security and compliance of Amazon SageMaker as part of multiple Amazon compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of Amazon services in scope of specific compliance programs, see [Amazon Services in Scope by Compliance Program](#). For general information, see [Amazon Compliance Programs](#).

You can download third-party audit reports using Amazon Artifact. For more information, see [Downloading Reports in Amazon Artifact](#).

Your compliance responsibility when using Amazon SageMaker is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. Amazon provides the following resources to help with compliance:

- [Security and Compliance Quick Start Guides](#) – These deployment guides discuss architectural considerations and provide steps for deploying security- and compliance-focused baseline environments on Amazon.
- [Architecting for HIPAA Security and Compliance Whitepaper](#) – This whitepaper describes how companies can use Amazon to create HIPAA-compliant applications.
- [Amazon Compliance Resources](#) – This collection of workbooks and guides might apply to your industry and location.
- [Amazon Config](#) – This Amazon service assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.
- [Amazon Security Hub](#) – This Amazon service provides a comprehensive view of your security state within Amazon that helps you check your compliance with security industry standards and best practices.

## Resilience in Amazon SageMaker

The Amazon global infrastructure is built around Amazon Regions and Availability Zones. Amazon Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between Availability Zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about Amazon Regions and Availability Zones, see [Amazon Global Infrastructure](#).

In addition to the Amazon global infrastructure, Amazon SageMaker offers several features to help support your data resiliency and backup needs.

## Infrastructure Security in Amazon SageMaker

As a managed service, Amazon SageMaker is protected by the Amazon global network security procedures that are described in the [Amazon Web Services: Overview of Security Processes](#) whitepaper.

You use Amazon published API calls to access Amazon SageMaker through the network. Clients must support Transport Layer Security (TLS) 1.0 or later. We recommend TLS 1.2 or later. Clients must also support cipher suites with perfect forward secrecy (PFS) such as Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Ephemeral Diffie-Hellman (ECDHE). Most modern systems such as Java 7 and later support these modes.

Additionally, requests must be signed by using an access key ID and a secret access key that is associated with an IAM principal. Or you can use the [Amazon Security Token Service](#) (Amazon STS) to generate temporary security credentials to sign requests.

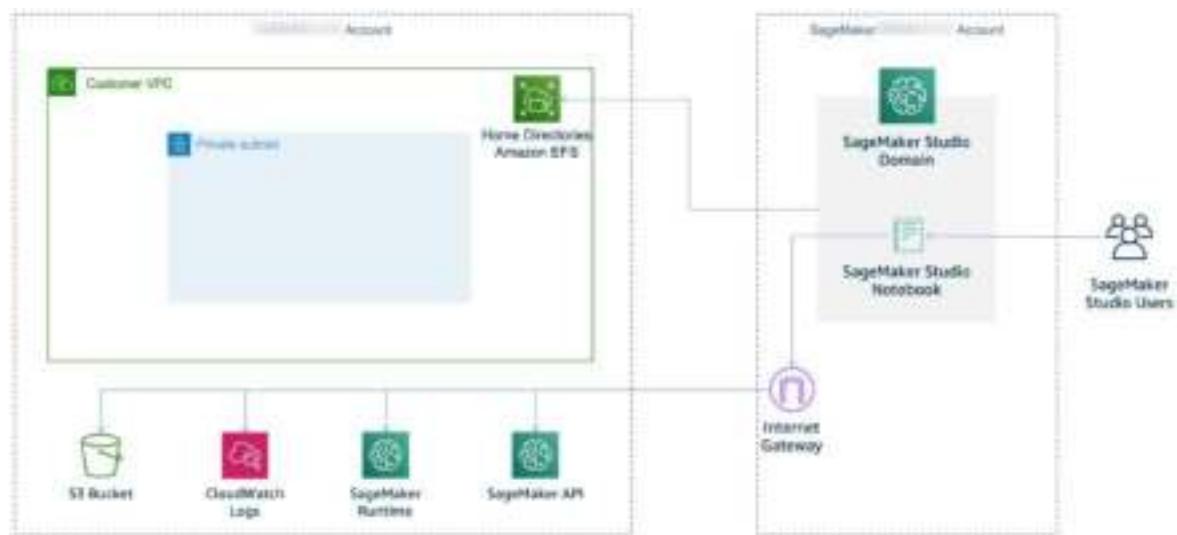
### Topics

- [Connect SageMaker Studio Notebooks to Resources in a VPC \(p. 2491\)](#)
- [Connect a Notebook Instance to Resources in a VPC \(p. 2493\)](#)
- [Training and Inference Containers Run in Internet-Free Mode \(p. 2494\)](#)
- [SageMaker Scans Amazon Web Services Marketplace Training and Inference Containers for Security Vulnerabilities \(p. 2495\)](#)
- [Connect to SageMaker Through a VPC Interface Endpoint \(p. 2495\)](#)
- [Give SageMaker Compilation Jobs Access to Resources in Your Amazon VPC \(p. 2504\)](#)
- [Give SageMaker Processing Jobs Access to Resources in Your Amazon VPC \(p. 2507\)](#)
- [Give SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC \(p. 2510\)](#)
- [Give SageMaker Training Jobs Access to Resources in Your Amazon VPC \(p. 2513\)](#)
- [Give Batch Transform Jobs Access to Resources in Your Amazon VPC \(p. 2516\)](#)
- [Give Amazon SageMaker Clarify Jobs Access to Resources in Your Amazon VPC \(p. 2519\)](#)

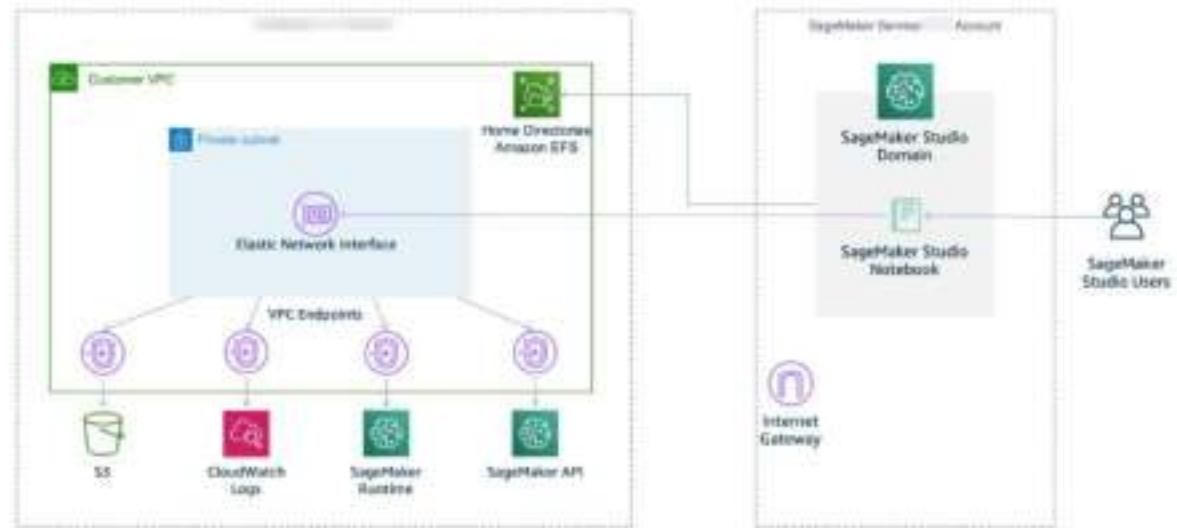
## Connect SageMaker Studio Notebooks to Resources in a VPC

Amazon SageMaker Studio allows direct internet access by default. This could provide an avenue for unauthorized access to your data. For example, if you install malicious code on your computer (in the form of a publicly available notebook or a publicly available source code library), it could access your data. You can choose to restrict which traffic can access the internet by launching Studio in a Virtual Private Cloud (VPC) of your choosing. This allows you fine-grained control of the network access and internet connectivity of your SageMaker Studio notebooks. You can disable direct internet access to add an additional layer of security.

By default, SageMaker Studio provides a network interface that allows communication with the internet through a VPC managed by SageMaker. Traffic to Amazon services like Amazon S3 and CloudWatch goes through an internet gateway as does traffic that accesses the SageMaker API and SageMaker runtime. Traffic between the domain and your Amazon EFS volume goes through the VPC that you specified when you onboarded to Studio or called the [CreateDomain](#) API. The following diagram shows the default configuration.



To disable direct internet access, you can specify the VPC-only network access type when you onboard to Studio or call the [CreateDomain](#) API. Doing so prevents SageMaker from providing internet access to your Studio notebooks. As a result, you won't be able to run a Studio notebook unless your VPC has an interface endpoint to the SageMaker API and runtime, or a NAT gateway with internet access, and your security groups allow outbound connections. The following diagram shows a configuration for using VPC-only mode.



### VPC requirements in VpcOnly

When you choose `VpcOnly`, your VPC must contain the following:

- Subnets with one IP address for each instance. For more information, see [VPC and subnet sizing for IPv4](#).

#### Note

You can configure only subnets with a default tenancy VPC in which your instance runs on shared hardware. For more information on the tenancy attribute for VPCs, see [Dedicated Instances](#).

- One or more security groups with inbound and outbound rules that together allow the following traffic:

- NFS traffic over TCP on port 2049 between the domain and the Amazon EFS volume.
- TCP traffic within the security group. This is required for connectivity between the JupyterServer app and the KernelGateway apps.
- If you want to allow internet access, you must use NAT gateway with access to internet (for example, via an internet gateway).
- If you don't want to allow internet access, you must create interface VPC endpoints (Amazon PrivateLink) to access the following:
  - The SageMaker API and SageMaker runtime. This is required to run Studio notebooks and to train and host models.
  - Amazon S3 and other Amazon services you require.
  - If you're using SageMaker Projects in SageMaker Studio without internet access, you need a VPC endpoint for Service Catalog.

You must associate the security groups for your VPC with these endpoints.

**Note**

For a customer working within VPC mode, company firewalls can cause connection issues with SageMaker Studio or between JupyterServer and the KernelGateway. Make the following checks if you encounter one of these issues when using SageMaker Studio from behind a firewall.

- Check that the Studio URL is in your networks allowlist.
- Check that the websocket connections are not blocked. Jupyter uses websocket under the hood. If the KernelGateway application is InService, JupyterServer may not be able to connect to the KernelGateway. You should see this problem when opening System Terminal as well.

**For more information**

- [Securing Amazon SageMaker Studio connectivity using a private VPC](#).
- [Security groups for your VPC](#)
- [Connect to SageMaker Through a VPC Interface Endpoint \(p. 2495\)](#)
- [VPC with public and private subnets \(NAT\)](#)

## Connect a Notebook Instance to Resources in a VPC

SageMaker notebook instances are internet-enabled by default. This allows you to download popular packages and notebooks, customize your development environment, and work efficiently. However, this could provide an additional avenue for unauthorized access to your data. For example, a malicious user or code that you accidentally install on the computer (in the form of a publicly available notebook or a publicly available source code library) could access your data. You can choose to launch your notebook instance in your Virtual Private Cloud (VPC) to restrict which traffic can go through the public Internet. When launched with your VPC attached, the notebook instance can either be configured with or without direct internet access.

When your notebook allows *direct internet access*, SageMaker provides a network interface that allows the notebook to communicate with the internet through a VPC managed by SageMaker. Traffic within your VPC's CIDR goes through elastic network interface created in your VPC. All the other traffic goes through the network interface created by SageMaker, which is essentially through the public internet. Traffic to gateway VPC endpoints like Amazon S3 and DynamoDB goes through the public internet, while traffic to interface VPC endpoints (interface endpoints) still goes through your VPC. If you want to use gateway VPC endpoints, you might want to disable direct internet access.

To disable direct internet access, you can specify a VPC for your notebook instance. By doing so, you prevent SageMaker from providing internet access to your notebook instance. As a result, the notebook

instance won't be able to train or host models unless your VPC has an interface endpoint (PrivateLink) or a NAT gateway, and your security groups allow outbound connections.

For information about creating a VPC interface endpoint to use Amazon PrivateLink for your notebook instance, see [Connect to a Notebook Instance Through a VPC Interface Endpoint \(p. 2497\)](#). For information about setting up a NAT gateway for your VPC, see [VPC with Public and Private Subnets \(NAT\)](#) in the *Amazon Virtual Private Cloud User Guide*. For information about security groups, see [Security Groups for Your VPC](#). For more information about networking configurations in each networking mode and configuring network on premise, see [Understanding Amazon SageMaker notebook instance networking configurations and advanced routing options](#).

## Security and Shared Notebook Instances

A SageMaker notebook instance is designed to work best for an individual user. It is designed to give data scientists and other users the most power for managing their development environment.

A notebook instance user has root access for installing packages and other pertinent software. We recommend that you exercise judgement when granting individuals access to notebook instances that are attached to a VPC that contains sensitive information. For example, you might grant a user access to a notebook instance with an IAM policy, as shown in the following example:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "sagemaker>CreatePresignedNotebookInstanceUrl",  
            "Resource": "arn:aws:sagemaker:region:account-id:notebook-instance/  
myNotebookInstance"  
        }  
    ]  
}
```

## Training and Inference Containers Run in Internet-Free Mode

SageMaker training and deployed inference containers are internet-enabled by default. This allows containers to access external services and resources on the public internet as part of your training and inference workloads. However, this could provide an avenue for unauthorized access to your data. For example, a malicious user or code that you accidentally install on the container (in the form of a publicly available source code library) could access your data and transfer it to a remote host.

If you use an Amazon VPC by specifying a value for the `VpcConfig` parameter when you call `CreateTrainingJob`, `CreateHyperParameterTuningJob`, or `CreateModel`, you can protect your data and resources by managing security groups and restricting internet access from your VPC. However, this comes at the cost of additional network configuration, and has the risk of configuring your network incorrectly. If you do not want SageMaker to provide external network access to your training or inference containers, you can enable network isolation when you create your training job or model by setting the value of the `EnableNetworkIsolation` parameter to `True` when you call `CreateTrainingJob`, `CreateHyperParameterTuningJob`, or `CreateModel`.

If you enable network isolation, the containers can't make any outbound network calls, even to other Amazon services such as Amazon S3. Additionally, no Amazon credentials are made available to the container runtime environment. In the case of a training job with multiple instances, network inbound and outbound traffic is limited to the peers of each training container. SageMaker still performs download and upload operations against Amazon S3 using your SageMaker execution role in isolation from the training or inference container.

Network isolation is required for training jobs and models run using resources from Amazon Web Services Marketplace. Network isolation can be used in conjunction with a VPC. In this scenario, download and upload of customer data and model artifacts are routed via your VPC subnet. However, the training and inference containers themselves continue to be isolated from the network, and do not have access to any resource within your VPC or on the internet.

Network isolation is not supported by the following managed SageMaker containers as they require access to Amazon S3:

- Chainer
- PyTorch
- Scikit-learn
- SageMaker Reinforcement Learning

## SageMaker Scans Amazon Web Services Marketplace Training and Inference Containers for Security Vulnerabilities

To meet our security requirements, algorithms and model packages listed in Amazon Web Services Marketplace are scanned for Common Vulnerabilities and Exposures (CVE). CVE is a list of publicly known information about security vulnerability and exposure. The National Vulnerability Database (NVD) provides CVE details such as severity, impact rating, and fix information. Both CVE and NVD are available for public consumption and free for security tools and services to use. For more information, see [http://cve.mitre.org/about/faqs.html#what\\_is\\_cve](http://cve.mitre.org/about/faqs.html#what_is_cve).

## Connect to SageMaker Through a VPC Interface Endpoint

You can connect directly to the SageMaker API or to the SageMaker Runtime through an [interface endpoint](#) in your Virtual Private Cloud (VPC) instead of connecting over the public internet. When you use a VPC interface endpoint, communication between your VPC and the SageMaker API or Runtime is conducted entirely and securely within the Amazon network.

The SageMaker API and Runtime support [Amazon Virtual Private Cloud](#) (Amazon VPC) interface endpoints that are powered by [Amazon PrivateLink](#). Each VPC endpoint is represented by one or more [Elastic Network Interfaces](#) with private IP addresses in your VPC subnets.

The VPC interface endpoint connects your VPC directly to the SageMaker API or Runtime without an internet gateway, NAT device, VPN connection, or Amazon Direct Connect connection. The instances in your VPC don't need public IP addresses to communicate with the SageMaker API or Runtime.

You can create an interface endpoint to connect to SageMaker or to SageMaker Runtime with either the Amazon console or Amazon Command Line Interface (Amazon CLI) commands. For instructions, see [Creating an Interface Endpoint](#).

*After you have created a VPC endpoint, you can use the following example CLI commands that use the endpoint-url parameter to specify interface endpoints to the SageMaker API or Runtime:*

```
aws sagemaker list-notebook-instances --endpoint-
url VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com

aws sagemaker list-training-jobs --endpoint-
url VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com
```

```
aws sagemaker-runtime invoke-endpoint --endpoint-
url VPC_Endpoint_ID.runtime.sagemaker.Region.vpce.amazonaws.com \
--endpoint-name Endpoint_Name \
--body "Endpoint_Body" \
--content-type "Content_Type" \
Output_File
```

If you enable private DNS hostnames for your VPC endpoint, you don't need to specify the endpoint URL. The SageMaker API DNS hostname that the CLI and SageMaker SDK use by default ([https://api.sagemaker.\*\*Region\*\*.amazonaws.com](https://api.sagemaker.<b>Region</b>.amazonaws.com)) resolves to your VPC endpoint. Similarly, the SageMaker Runtime DNS hostname that the CLI and SageMaker Runtime SDK use by default ([https://runtime.sagemaker.\*\*Region\*\*.amazonaws.com](https://runtime.sagemaker.<b>Region</b>.amazonaws.com)) resolves to your VPC endpoint.

The SageMaker API and Runtime support VPC endpoints in all Amazon Regions where both [Amazon VPC](#) and [SageMaker](#) are available. SageMaker supports making calls to all of its [Operations](#) inside your VPC. The result `AuthorizedUrl` from the `CreatePresignedNotebookInstanceUrl` is not supported by Private Link. For information about how to enable PrivateLink for the authorized URL that users use to connect to a notebook instance, see [Connect to a Notebook Instance Through a VPC Interface Endpoint \(p. 2497\)](#).

To learn more about Amazon PrivateLink, see the [Amazon PrivateLink documentation](#). Refer to [VPC Pricing](#) for the price of VPC Endpoints. To learn more about VPC and Endpoints, see [Amazon VPC](#). For information about how to use identity-based Amazon Identity and Access Management policies to restrict access to the SageMaker API and runtime, see [Control Access to the SageMaker API by Using Identity-based Policies \(p. 2441\)](#).

## Create a VPC Endpoint Policy for SageMaker

You can create a policy for Amazon VPC endpoints for SageMaker to specify the following:

- The principal that can perform actions.
- The actions that can be performed.
- The resources on which actions can be performed.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

### Note

VPC endpoint policies aren't supported for Federal Information Processing Standard (FIPS) SageMaker runtime endpoints for `runtime_InvokeEndpoint`.

The following example VPC endpoint policy specifies that all users who have access to the VPC interface endpoint are allowed to invoke the SageMaker hosted endpoint named `myEndpoint`.

```
{
  "Statement": [
    {
      "Action": "sagemaker:InvokeEndpoint",
      "Effect": "Allow",
      "Resource": "arn:aws:sagemaker:us-west-2:123456789012:endpoint/myEndpoint",
      "Principal": "*"
    }
  ]
}
```

In this example, the following are denied:

- Other SageMaker API actions, such as `sagemaker:CreateEndpoint` and `sagemaker:CreateTrainingJob`.

- Invoking SageMaker hosted endpoints other than `myEndpoint`.

**Note**

In this example, users can still take other SageMaker API actions from outside the VPC. For information about how to restrict API calls to those from within the VPC, see [Control Access to the SageMaker API by Using Identity-based Policies \(p. 2441\)](#).

## Create a VPC Endpoint Policy for SageMaker Feature Store

To create a VPC Endpoint for Feature Store, use the following endpoint template substituting your `VPC_Endpoint_ID.api` and `Region`:

`VPC_Endpoint_ID.api.featurestore-runtime.sagemaker.Region.vpce.amazonaws.com`

## Connect to a Notebook Instance Through a VPC Interface Endpoint

You can connect to your notebook instance from your VPC through an [interface endpoint](#) in your Virtual Private Cloud (VPC) instead of connecting over the public internet. When you use a VPC interface endpoint, communication between your VPC and the notebook instance is conducted entirely and securely within the Amazon network.

SageMaker notebook instances support [Amazon Virtual Private Cloud](#) (Amazon VPC) interface endpoints that are powered by [Amazon PrivateLink](#). Each VPC endpoint is represented by one or more [Elastic Network Interfaces](#) with private IP addresses in your VPC subnets.

**Note**

Before you create an interface VPC endpoint to connect to a notebook instance, create an interface VPC endpoint to connect to the SageMaker API. That way, when users call `CreatePresignedNotebookInstanceUrl` to get the URL to connect to the notebook instance, that call also goes through the interface VPC endpoint. For information, see [Connect to SageMaker Through a VPC Interface Endpoint \(p. 2495\)](#).

You can create an interface endpoint to connect to your notebook instance with either the Amazon console or Amazon Command Line Interface (Amazon CLI) commands. For instructions, see [Creating an Interface Endpoint](#). Make sure that you create an interface endpoint for all of the subnets in your VPC from which you want to connect to the notebook instance.

When you create the interface endpoint, specify `aws.sagemaker.region.notebook` as the service name. After you create a VPC endpoint, enable private DNS for your VPC endpoint. Anyone using the SageMaker API, the Amazon CLI, or the console to connect to the notebook instance from within the VPC will connect to the notebook instance through the VPC endpoint instead of the public internet.

SageMaker notebook instances support VPC endpoints in all Amazon Regions where both [Amazon VPC](#) and [SageMaker](#) are available.

### Topics

- [Connect Your Private Network to Your VPC \(p. 2497\)](#)
- [Create a VPC Endpoint Policy for SageMaker Notebook Instances \(p. 2498\)](#)
- [Restrict Access to Connections from Within Your VPC \(p. 2498\)](#)

## Connect Your Private Network to Your VPC

To connect to your notebook instance through your VPC, you either have to connect from an instance that is inside the VPC, or connect your private network to your VPC by using an Amazon Virtual Private

Network (VPN) or Amazon Direct Connect. For information about Amazon VPN, see [VPN Connections in the \*Amazon Virtual Private Cloud User Guide\*](#). For information about Amazon Direct Connect, see [Creating a Connection in the \*Amazon Direct Connect User Guide\*](#).

## Create a VPC Endpoint Policy for SageMaker Notebook Instances

You can create a policy for Amazon VPC endpoints for SageMaker notebook instances to specify the following:

- The principal that can perform actions.
  - The actions that can be performed.
  - The resources on which actions can be performed.

For more information, see [Controlling Access to Services with VPC Endpoints](#) in the *Amazon VPC User Guide*.

The following example of a VPC endpoint policy specifies that all users that have access to the endpoint are allowed to access the notebook instance named `myNotebookInstance`.

```
{  
    "Statement": [  
        {  
            "Action": "sagemaker>CreatePresignedNotebookInstanceUrl",  
            "Effect": "Allow",  
            "Resource": "arn:aws:sagemaker:us-west-2:123456789012:notebook-instance/  
myNotebookInstance",  
            "Principal": "*"  
        }  
    ]  
}
```

Access to other notebook instances is denied.

## Restrict Access to Connections from Within Your VPC

Even if you set up an interface endpoint in your VPC, individuals outside the VPC can connect to the notebook instance over the internet.

## **Important**

If you apply an IAM policy similar to one of the following, users can't access the specified SageMaker APIs or the notebook instance through the console.

To restrict access to only connections made from within your VPC, create an Amazon Identity and Access Management policy that restricts access to only calls that come from within your VPC. Then add that policy to every Amazon Identity and Access Management user, group, or role used to access the notebook instance.

## Note

**Note** This policy allows connections only to callers within a subnet where you created an interface endpoint.

```
{
    "Id": "notebook-example-1",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable Notebook Access",
            "Effect": "Allow",
            "Action": [
                "sagemaker>CreatePresignedNotebookInstanceUrl"
            ],
            "Resource": [
                "arn:aws:sagemaker:::notebook-instance/*"
            ]
        }
    ]
}
```

```

        "sagemaker:DescribeNotebookInstance"
    ],
    "Resource": "*",
    "Condition": {
        "StringEquals": {
            "aws:SourceVpc": "vpc-111bbaaa"
        }
    }
}
]
}
}

```

If you want to restrict access to the notebook instance to only connections made using the interface endpoint, use the `aws:SourceVpce` condition key instead of `aws:SourceVpc`:

```

{
    "Id": "notebook-example-1",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable Notebook Access",
            "Effect": "Allow",
            "Action": [
                "sagemaker>CreatePresignedNotebookInstanceUrl",
                "sagemaker:DescribeNotebookInstance"
            ],
            "Resource": "*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:sourceVpce": [
                        "vpce-111bbccc",
                        "vpce-111bbddd"
                    ]
                }
            }
        }
    ]
}

```

Both of these policy examples assume that you have also created an interface endpoint for the SageMaker API. For more information, see [Connect to SageMaker Through a VPC Interface Endpoint \(p. 2495\)](#). In the second example, one of the values for `aws:SourceVpce` is the ID of the interface endpoint for the notebook instance. The other is the ID of the interface endpoint for the SageMaker API.

The policy examples here include `DescribeNotebookInstance` because typically you would call `DescribeNotebookInstance` to make sure that the `NotebookInstanceStatus` is `InService` before you try to connect to it. For example:

```

aws sagemaker describe-notebook-instance \
    --notebook-instance-name myNotebookInstance

{
    "NotebookInstanceArn": "arn:aws:sagemaker:us-west-2:1234567890ab:notebook-instance/mynotebookinstance",
    "NotebookInstanceName": "myNotebookInstance",
    "NotebookInstanceStatus": "InService",
    "Url": "mynotebookinstance.notebook.us-west-2.sagemaker.aws",
    "InstanceType": "ml.m4.xlarge",
    "RoleArn": "

```

```
"arn:aws:iam::1234567890ab:role/service-role/AmazonSageMaker-  
ExecutionRole-12345678T123456",  
    "LastModifiedTime": 1540334777.501,  
    "CreationTime": 1523050674.078,  
    "DirectInternetAccess": "Disabled"  
}  
aws sagemaker create-presigned-notebook-instance-url --notebook-instance-name  
myNotebookInstance  
  
{  
    "AuthorizedUrl": "https://mynotebookinstance.notebook.us-west-2.sagemaker.aws?  
authToken=AuthToken  
}
```

For both of these calls, if you did not enable private DNS hostnames for your VPC endpoint, or if you are using a version of the Amazon SDK that was released before August 13, 2018, you must specify the endpoint URL in the call. For example, the call to `create-presigned-notebook-instance-url` would be:

```
aws sagemaker create-presigned-notebook-instance-url  
--notebook-instance-name myNotebookInstance --endpoint-url  
VPC_Endpoint_ID.api.sagemaker.Region.vpce.amazonaws.com
```

## Connect Your Private Network to Your VPC

To call the SageMaker API and runtime through your VPC, you have to connect from an instance that is inside the VPC or connect your private network to your VPC by using an Amazon Virtual Private Network (VPN) or Amazon Direct Connect. For information about Amazon VPN, see [VPN Connections](#) in the *Amazon Virtual Private Cloud User Guide*. For information about Amazon Direct Connect, see [Creating a Connection](#) in the *Amazon Direct Connect User Guide*.

## Connect to SageMaker Studio Through an Interface VPC Endpoint

You can connect to Amazon SageMaker Studio from your [Amazon Virtual Private Cloud](#) (Amazon VPC) through an [interface endpoint](#) in your VPC instead of connecting over the internet. When you use an interface VPC endpoint (interface endpoint), communication between your VPC and Studio is conducted entirely and securely within the Amazon network.

SageMaker Studio supports interface endpoints that are powered by [Amazon PrivateLink](#). Each interface endpoint is represented by one or more [Elastic network interfaces](#) with private IP addresses in your VPC subnets.

You can create an interface endpoint to connect to Studio with either the Amazon console or the Amazon Command Line Interface (Amazon CLI). For instructions, see [Creating an interface endpoint](#). Make sure that you create interface endpoints for all of the subnets in your VPC from which you want to connect to Studio.

When you create an interface endpoint, ensure that the security groups on your endpoint allow inbound access for HTTPS traffic from the security groups associated with SageMaker Studio. For more information, see [Control access to services with VPC endpoints](#).

### Note

In addition to creating an interface endpoint to connect to SageMaker Studio, create an interface endpoint to connect to the Amazon SageMaker API. When users call `CreatePresignedDomainUrl` to get the URL to connect to Studio, that call goes through the interface endpoint used to connect to the SageMaker API.

When you create the interface endpoint, specify `aws.sagemaker.Region.studio` as the service name. After you create the interface endpoint, enable private DNS for your endpoint. Anyone who uses the SageMaker API, the Amazon CLI, or the console to connect to SageMaker Studio from within the VPC will connect to Studio through the interface endpoint instead of the public internet. You also need to setup a custom DNS with Private Hosted Zones for the Amazon VPC endpoint so SageMaker Studio can access the SageMaker API using the `api.sagemaker.$region.amazonaws.com` endpoint rather than using the VPC Endpoint URL. For instructions on setting up a Private Hosted Zone, see [Working with private hosted zones](#).

Studio supports interface endpoints in all Amazon Regions where both [Amazon SageMaker](#) and [Amazon VPC](#) are available.

### Topics

- [Create a VPC Endpoint Policy for SageMaker Studio \(p. 2501\)](#)
- [Allow Access Only from Within Your VPC \(p. 2502\)](#)

## Create a VPC Endpoint Policy for SageMaker Studio

You can attach an Amazon VPC endpoint policy to the interface VPC endpoints that you use to connect to SageMaker Studio. The endpoint policy controls access to Studio. You can specify the following:

- The principal that can perform actions
- The actions that can be performed
- The resources on which actions can be performed

To use a VPC endpoint with SageMaker Studio, your endpoint policy must allow the `CreateApp` operation on the `KernelGateway` app type. This allows traffic that is routed to through the VPC endpoint to call the `CreateApp` API. The following example VPC endpoint policy shows how to allow the `CreateApp` operation.

```
{  
  "Statement": [  
    {  
      "Action": "sagemaker:CreateApp",  
      "Effect": "Allow",  
      "Resource": "arn:aws:sagemaker:us-west-2:acct-id:user-profile/domain-id/*",  
      "Principal": "*"  
    }  
  ]  
}
```

For more information, see [Controlling access to services with VPC endpoints](#).

The following example of a VPC endpoint policy specifies that all users that have access to the endpoint are allowed to access the user profiles in the SageMaker Studio domain with the specified domain ID. Access to other domains is denied.

```
{  
  "Statement": [  
    {  
      "Action": "sagemaker:CreatePresignedDomainUrl",  
      "Effect": "Allow",  
      "Resource": "arn:aws:sagemaker:us-west-2:acct-id:user-profile/domain-id/*",  
      "Principal": "*"  
    }  
  ]  
}
```

## Allow Access Only from Within Your VPC

Users outside your VPC can connect to SageMaker Studio over the internet even if you set up an interface endpoint in your VPC.

To allow access to only connections made from within your VPC, create an Amazon Identity and Access Management (IAM) policy to that effect. Add that policy to every IAM user, group, or role used to access Studio. This feature is only supported in IAM mode, and is not supported in SSO mode. The following examples demonstrate how to create such policies.

### Important

If you apply an IAM policy similar to one of the following examples, users can't access SageMaker Studio or the specified SageMaker APIs through the SageMaker console. To access Studio, users must use a presigned URL or call the SageMaker APIs directly.

#### Example 1

The following policy allows connections only to callers within the subnet where you created the interface endpoint.

```
{  
    "Id": "sagemaker-studio-example-1",  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Enable SageMaker Studio Access",  
            "Effect": "Allow",  
            "Action": [  
                "sagemaker>CreatePresignedDomainUrl",  
                "sagemaker:DescribeUserProfile"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "StringEquals": {  
                    "aws:SourceVpc": "vpc-111bbaaa"  
                }  
            }  
        }  
    ]  
}
```

#### Example 2

The following policy allows connections only to those made through the interface endpoints specified by the `aws:sourceVpc` condition key. For example, the first interface endpoint could allow access through the SageMaker Studio Control Panel. The second interface endpoint could allow access through the SageMaker API.

```
{  
    "Id": "sagemaker-studio-example-2",  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Enable SageMaker Studio Access",  
            "Effect": "Allow",  
            "Action": [  
                "sagemaker>CreatePresignedDomainUrl",  
                "sagemaker:DescribeUserProfile"  
            ],  
            "Resource": "*",  
            "Condition": {  
                "ForAnyValue:StringEquals": {  
                    "aws:sourceVpc": "vpc-111bbaaa"  
                }  
            }  
        }  
    ]  
}
```

```

        "aws:sourceVpce": [
            "vpce-111bbccc",
            "vpce-111bbddd"
        ]
    }
}
]
}
}

```

This policy includes the `DescribeUserProfile` action. Typically you call `DescribeUserProfile` to make sure that the status of the user profile is `InService` before you try to connect to the domain. For example:

```
aws sagemaker describe-user-profile \
--domain-id domain-id \
--user-profile-name profile-name
```

**Response:**

```
{
    "DomainId": "domain-id",
    "UserProfileArn": "arn:aws:sagemaker:us-west-2:acct-id:user-profile/domain-id/profile-
name",
    "UserProfileName": "profile-name",
    "HomeEfsFileSystemUid": "200001",
    "Status": "InService",
    "LastModifiedTime": 1605418785.555,
    "CreationTime": 1605418477.297
}
```

```
aws sagemaker create-presigned-domain-url
--domain-id domain-id \
--user-profile-name profile-name
```

**Response:**

```
{
    "AuthorizedUrl": "https://domain-id.studio.us-west-2.sagemaker.aws/auth?
token=AuthToken"
}
```

For both of these calls, if you are using a version of the Amazon SDK that was released before August 13, 2018, you must specify the endpoint URL in the call. For example, the call to `create-presigned-domain-url` would be:

```
aws sagemaker create-presigned-domain-url
--domain-id domain-id \
--user-profile-name profile-name \
--endpoint-url vpc-endpoint-id.api.sagemaker.Region.vpce.amazonaws.com
```

### Example 3

The following policy allows connections only from the specified range of IP addresses using the `aws:SourceIp` condition key.

```
{
    "Id": "sagemaker-studio-example-3",
```

```
"Version": "2012-10-17",
"Statement": [
    {
        "Sid": "Enable SageMaker Studio Access",
        "Effect": "Allow",
        "Action": [
            "sagemaker>CreatePresignedDomainUrl",
            "sagemaker:DescribeUserProfile"
        ],
        "Resource": "*",
        "Condition": {
            "IpAddress": {
                "aws:SourceIp": [
                    "192.0.2.0/24",
                    "203.0.113.0/24"
                ]
            }
        }
    }
]
```

#### Example 4

If you are accessing SageMaker Studio through an interface endpoint, you can use the `aws:VpcSourceIp` condition key to allow connections only from the specified range of IP addresses as shown in the following policy.

```
{
    "Id": "sagemaker-studio-example-4",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Enable SageMaker Studio Access",
            "Effect": "Allow",
            "Action": [
                "sagemaker>CreatePresignedDomainUrl",
                "sagemaker:DescribeUserProfile"
            ],
            "Resource": "*",
            "Condition": {
                "IpAddress": {
                    "aws:VpcSourceIp": [
                        "192.0.2.0/24",
                        "203.0.113.0/24"
                    ]
                }
            }
        }
    ]
}
```

## Give SageMaker Compilation Jobs Access to Resources in Your Amazon VPC

SageMaker Neo runs compilation jobs in an Amazon Virtual Private Cloud by default. However, compilation jobs access Amazon resources—such as the Amazon Simple Storage Service (Amazon S3) buckets where you store model artifacts—over the internet.

To control access to your data, we recommend that you create a private VPC and configure it so that they aren't accessible over the internet. For information about creating and configuring a VPC, see

[Getting Started With Amazon VPC](#) in the *Amazon VPC User Guide*. Using a VPC helps to protect your data because you can configure your VPC so that it is not connected to the internet. For more information, see [VPC Flow Logs](#) in the *Amazon VPC User Guide*.

You specify your private VPC configuration when you create compilation jobs by specifying subnets and security groups. When you specify the subnets and security groups, SageMaker creates *elastic network interfaces* that are associated with your security groups in one of the subnets. Network interfaces allow SageMaker Neo's compilation jobs to connect to resources in your VPC. For information about network interfaces, see [Elastic Network Interfaces](#) in the *Amazon VPC User Guide*.

**Note**

For compilation jobs, you can configure only subnets with a default tenancy VPC in which your job runs on shared hardware. For more information on the tenancy attribute for VPCs, see [Dedicated Instances](#).

## Configure a Compilation Job for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the `VpcConfig` request parameter of the [CreateCompilationJob](#) API, or provide this information when you create a compilation job in the SageMaker console. SageMaker Neo uses this information to create network interfaces and attach them to your compilation jobs. The network interfaces provide compilation jobs with a network connection within your VPC that is not connected to the internet. They also enable your compilation job to connect to resources in your private VPC. The following is an example of the `VpcConfig` parameter that you include in your call to `CreateCompilationJob`:

```
vpcConfig: { "Subnets": [  
    "subnet-0123456789abcdef0",  
    "subnet-0123456789abcdef1",  
    "subnet-0123456789abcdef2"  
],  
  "SecurityGroupIds": [  
    "sg-0123456789abcdef0"  
  ]  
}
```

## Configure Your Private VPC for SageMaker Compilation

When configuring the private VPC for your SageMaker compilation jobs, use the following guidelines. For information about setting up a VPC, see [Working with VPCs and Subnets](#) in the *Amazon VPC User Guide*.

### Topics

- [Ensure That Subnets Have Enough IP Addresses \(p. 2505\)](#)
- [Create an Amazon S3 VPC Endpoint \(p. 2505\)](#)
- [Use a Custom Endpoint Policy to Restrict Access to S3 \(p. 2506\)](#)
- [Configure Route Tables \(p. 2507\)](#)
- [Configure the VPC Security Group \(p. 2507\)](#)

### Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each instance in a compilation job. For more information, see [VPC and Subnet Sizing for IPv4](#) in the *Amazon VPC User Guide*.

### Create an Amazon S3 VPC Endpoint

If you configure your VPC to block access to the internet, SageMaker Neo can't connect to the Amazon S3 buckets that contain your models unless you create a VPC endpoint that allows access. By creating a VPC

endpoint, you allow your SageMaker Neo compilation jobs to access the buckets where you store your data and model artifacts . We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see [Endpoints for Amazon S3](#).

#### To create an S3 VPC endpoint:

1. Open the Amazon VPC console at <https://console.amazonaws.cn/vpc/>.
2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**
3. For **Service Name**, search for **com.amazonaws.region.s3**, where **region** is the name of the region where your VPC resides.
4. Choose the **Gateway** type.
5. For **VPC**, choose the VPC you want to use for this endpoint.
6. For **Configure route tables**, select the route tables to be used by the endpoint. The VPC service automatically adds a route to each route table you select that points any S3 traffic to the new endpoint.
7. For **Policy**, choose **Full Access** to allow full access to the S3 service by any user or service within the VPC. Choose **Custom** to restrict access further. For information, see [Use a Custom Endpoint Policy to Restrict Access to S3 \(p. 2515\)](#).

#### Use a Custom Endpoint Policy to Restrict Access to S3

The default endpoint policy allows full access to S3 for any user or service in your VPC. To further restrict access to S3, create a custom endpoint policy. For more information, see [Using Endpoint Policies for Amazon S3](#). You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see [Using Amazon S3 Bucket Policies](#). The following is a sample customized policy:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Principal": {
                "AWS": "*"
            },
            "Action": "s3:GetObject",
            "Resource": [
                "arn:aws:s3:::your-sample-bucket",
                "arn:aws:s3:::your-sample-bucket/*"
            ],
            "Condition": {
                "StringNotEquals": {
                    "aws:SourceVpce": [
                        "vpce-01234567890123456"
                    ]
                }
            }
        }
    ]
}
```

#### Add Permissions for Compilation Job Running in a Amazon VPC to Custom IAM Policies

The `SageMakerFullAccess` managed policy includes the permissions that you need to use models configured for Amazon VPC access with an endpoint. These permissions allow SageMaker Neo to create an elastic network interface and attach it to compilation job running in a Amazon VPC. If you use your own IAM policy, you must add the following permissions to that policy to use models configured for Amazon VPC access.

```
{"Version": "2012-10-17",
"Statement": [
    {"Effect": "Allow",
        "Action": [
            "ec2:DescribeVpcEndpoints",
            "ec2:DescribeDhcpOptions",
            "ec2:DescribeVpcs",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DeleteNetworkInterfacePermission",
            "ec2:DeleteNetworkInterface",
            "ec2:CreateNetworkInterfacePermission",
            "ec2:CreateNetworkInterface",
            "ec2:ModifyNetworkInterfaceAttribute"
        ],
        "Resource": "*"
    }
]
```

For more information about the `SageMakerFullAccess` managed policy, see [AmazonSageMakerFullAccess \(p. 2465\)](#).

## Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, `http://s3-aws-region.amazonaws.com/MyBucket`) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your compilation jobs resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see [Routing for Gateway Endpoints](#) in the *Amazon VPC User Guide*.

## Configure the VPC Security Group

In your security group for the compilation job, you must allow outbound communication to your Amazon S3 Amazon VPC endpoints and the subnet CIDR ranges used for the compilation job. For information, see [Security Group Rules](#) and [Control access to services with Amazon VPC endpoints](#).

# Give SageMaker Processing Jobs Access to Resources in Your Amazon VPC

SageMaker runs processing jobs in an Amazon Virtual Private Cloud by default. However, processing containers access Amazon resources—such as the Amazon S3 buckets where you store data—over the internet.

To control access to your data and processing containers, we recommend that you create a private VPC and configure it so that they aren't accessible over the internet. For information about creating and configuring a VPC, see [Getting Started With Amazon VPC](#) in the *Amazon VPC User Guide*. Using a VPC helps to protect your processing containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your processing containers by using VPC flow logs. For more information, see [VPC Flow Logs](#) in the *Amazon VPC User Guide*.

You specify your private VPC configuration when you create processing jobs by specifying subnets and security groups. When you specify the subnets and security groups, SageMaker creates *elastic network interfaces* that are associated with your security groups in one of the subnets. Network interfaces allow your processing containers to connect to resources in your VPC. For information about network interfaces, see [Elastic Network Interfaces](#) in the *Amazon VPC User Guide*.

## Configure a Processing Job for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the `NetworkConfig.VpcConfig` request parameter of the [CreateProcessingJob](#) API, or provide this information when you create a processing job in the SageMaker console. SageMaker uses this information to create network interfaces and attach them to your processing containers. The network interfaces provide your processing containers with a network connection within your VPC that is not connected to the internet. They also enable your processing job to connect to resources in your private VPC.

The following is an example of the `VpcConfig` parameter that you include in your call to `CreateProcessingJob`:

```
VpcConfig: {  
    "Subnets": [  
        "subnet-0123456789abcdef0",  
        "subnet-0123456789abcdef1",  
        "subnet-0123456789abcdef2"  
    ],  
    "SecurityGroupIds": [  
        "sg-0123456789abcdef0"  
    ]  
}
```

## Configure Your Private VPC for SageMaker Processing

When configuring the private VPC for your SageMaker processing jobs, use the following guidelines. For information about setting up a VPC, see [Working with VPCs and Subnets](#) in the *Amazon VPC User Guide*.

### Topics

- [Ensure That Subnets Have Enough IP Addresses \(p. 2508\)](#)
- [Create an Amazon S3 VPC Endpoint \(p. 2508\)](#)
- [Use a Custom Endpoint Policy to Restrict Access to S3 \(p. 2509\)](#)
- [Configure Route Tables \(p. 2509\)](#)
- [Configure the VPC Security Group \(p. 2510\)](#)
- [Connect to Resources Outside Your VPC \(p. 2510\)](#)

### Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each instance in a processing job. For more information, see [VPC and Subnet Sizing for IPv4](#) in the *Amazon VPC User Guide*.

### Create an Amazon S3 VPC Endpoint

If you configure your VPC so that processing containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your processing containers to access the buckets where you store your data. We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see [Endpoints for Amazon S3](#).

#### To create an S3 VPC endpoint:

1. Open the Amazon VPC console at <https://console.amazonaws.cn/vpc/>.
2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**
3. For **Service Name**, choose **com.amazonaws.*region*.s3**, where *region* is the name of the region where your VPC resides.

4. For **VPC**, choose the VPC you want to use for this endpoint.
5. For **Configure route tables**, select the route tables to be used by the endpoint. The VPC service automatically adds a route to each route table you select that points any S3 traffic to the new endpoint.
6. For **Policy**, choose **Full Access** to allow full access to the S3 service by any user or service within the VPC. Choose **Custom** to restrict access further. For information, see [Use a Custom Endpoint Policy to Restrict Access to S3 \(p. 2509\)](#).

## Use a Custom Endpoint Policy to Restrict Access to S3

The default endpoint policy allows full access to S3 for any user or service in your VPC. To further restrict access to S3, create a custom endpoint policy. For more information, see [Using Endpoint Policies for Amazon S3](#). You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see [Using Amazon S3 Bucket Policies](#).

### Restrict Package Installation on the Processing Container

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the processing container. If you don't want users to install packages from that repository, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```
{  
    "Statement": [  
        {  
            "Sid": "AmazonLinuxAMIRRepositoryAccess",  
            "Principal": "*",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Effect": "Deny",  
            "Resource": [  
                "arn:aws:s3:::packages.*.amazonaws.com/*",  
                "arn:aws:s3:::repo.*.amazonaws.com/*"  
            ]  
        }  
    ]  
}  
  
{  
    "Statement": [  
        { "Sid": "AmazonLinux2AMIRRepositoryAccess",  
          "Principal": "*",  
          "Action": [  
              "s3:GetObject"  
          ],  
          "Effect": "Deny",  
          "Resource": [  
              "arn:aws:s3:::amazonlinux.*.amazonaws.com/*"  
          ]  
        }  
    ]  
}
```

## Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, `http://s3-aws-region.amazonaws.com/MyBucket`) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your processing jobs resolve by

configuring the endpoint route tables. For information about VPC endpoint route tables, see [Routing for Gateway Endpoints](#) in the *Amazon VPC User Guide*.

## Configure the VPC Security Group

In distributed processing, you must allow communication between the different containers in the same processing job. To do that, configure a rule for your security group that allows inbound connections between members of the same security group. For information, see [Security Group Rules](#).

## Connect to Resources Outside Your VPC

If you configure your VPC so that it doesn't have internet access, processing jobs that use that VPC do not have access to resources outside your VPC. If your processing job needs access to resources outside your VPC, provide access with one of the following options:

- If your processing job needs access to an Amazon service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see [VPC Endpoints](#) in the *Amazon VPC User Guide*. For information about creating an interface VPC endpoint, see [Interface VPC Endpoints \(Amazon PrivateLink\)](#) in the *Amazon VPC User Guide*.
- If your processing job needs access to an Amazon service that doesn't support interface VPC endpoints or to a resource outside of Amazon, create a NAT gateway and configure your security groups to allow outbound connections. For information about setting up a NAT gateway for your VPC, see [Scenario 2: VPC with Public and Private Subnets \(NAT\)](#) in the *Amazon Virtual Private Cloud User Guide*.

## Give SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC

SageMaker hosts models in an Amazon Virtual Private Cloud by default. However, models access Amazon resources—such as the Amazon S3 buckets where you store training data and model artifacts—over the internet.

To avoid making your data and model containers accessible over the internet, we recommend that you create a private VPC and configure it to control access to them. For information about creating and configuring a VPC, see [Getting Started With Amazon VPC](#) in the *Amazon VPC User Guide*. Using a VPC helps to protect your training containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your training containers by using VPC flow logs. For more information, see [VPC Flow Logs](#) in the *Amazon VPC User Guide*.

You specify your private VPC configuration when you create a model by specifying subnets and security groups. When you specify the subnets and security groups, SageMaker creates *elastic network interfaces* that are associated with your security groups in one of the subnets. Network interfaces allow your model containers to connect to resources in your VPC. For information about network interfaces, see [Elastic Network Interfaces](#) in the *Amazon VPC User Guide*.

## Configure a Model for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the `vpcConfig` request parameter of the [CreateModel](#) API, or provide this information when you create a model in the SageMaker console. SageMaker uses this information to create network interfaces and attach them to your model containers. The network interfaces provide your model containers with a network connection within your VPC that is not connected to the internet. They also enable your model to connect to resources in your private VPC.

### Note

You must create at least two subnets in different availability zones in your private VPC, even if you have only one hosting instance.

The following is an example of the `VpcConfig` parameter that you include in your call to `CreateModel`:

```
vpcConfig: {  
    "Subnets": [  
        "subnet-0123456789abcdef0",  
        "subnet-0123456789abcdef1",  
        "subnet-0123456789abcdef2"  
    ],  
    "SecurityGroupIds": [  
        "sg-0123456789abcdef0"  
    ]  
}
```

## Configure Your Private VPC for SageMaker Hosting

When configuring the private VPC for your SageMaker models, use the following guidelines. For information about setting up a VPC, see [Working with VPCs and Subnets in the Amazon VPC User Guide](#).

### Topics

- [Ensure That Subnets Have Enough IP Addresses \(p. 2511\)](#)
- [Create an Amazon S3 VPC Endpoint \(p. 2511\)](#)
- [Use a Custom Endpoint Policy to Restrict Access to Amazon S3 \(p. 2512\)](#)
- [Add Permissions for Endpoint Access for Containers Running in a VPC to Custom IAM Policies \(p. 2512\)](#)
- [Configure Route Tables \(p. 2513\)](#)
- [Connect to Resources Outside Your VPC \(p. 2513\)](#)

## Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each model instance. For more information, see [VPC and Subnet Sizing for IPv4](#) in the *Amazon VPC User Guide*.

## Create an Amazon S3 VPC Endpoint

If you configure your VPC so that model containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your model containers to access the buckets where you store your data and model artifacts. We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see [Endpoints for Amazon S3](#).

### To create an Amazon S3 VPC endpoint:

1. Open the Amazon VPC console at <https://console.amazonaws.cn/vpc/>.
2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**
3. For **Service Name**, choose **com.amazonaws.*region*.s3**, where *region* is the name of the Amazon Region where your VPC resides.
4. For **VPC**, choose the VPC that you want to use for this endpoint.
5. For **Configure route tables**, choose the route tables that the endpoint will use. The VPC service automatically adds a route to each route table that you choose that points Amazon S3 traffic to the new endpoint.
6. For **Policy**, choose **Full Access** to allow full access to the Amazon S3 service by any user or service within the VPC. To restrict access further, choose **Custom**. For more information, see [Use a Custom Endpoint Policy to Restrict Access to Amazon S3 \(p. 2512\)](#).

## Use a Custom Endpoint Policy to Restrict Access to Amazon S3

The default endpoint policy allows full access to Amazon Simple Storage Service (Amazon S3) for any user or service in your VPC. To further restrict access to Amazon S3, create a custom endpoint policy. For more information, see [Using Endpoint Policies for Amazon S3](#).

You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see [Using Amazon S3 Bucket Policies](#).

### Restrict Package Installation on the Model Container with a Custom Endpoint Policy

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the model container. If you don't want users to install packages from those repositories, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```
{  
    "Statement": [  
        {  
            "Sid": "AmazonLinuxAMIRepositoryAccess",  
            "Principal": "*",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Effect": "Deny",  
            "Resource": [  
                "arn:aws:s3:::packages.*.amazonaws.com/*",  
                "arn:aws:s3:::repo.*.amazonaws.com/*"  
            ]  
        }  
    ]  
}  
  
{  
    "Statement": [  
        { "Sid": "AmazonLinux2AMIRepositoryAccess",  
          "Principal": "*",  
          "Action": [  
              "s3:GetObject"  
          ],  
          "Effect": "Deny",  
          "Resource": [  
              "arn:aws:s3:::amazonlinux.*.amazonaws.com/*"  
          ]  
        }  
    ]  
}
```

## Add Permissions for Endpoint Access for Containers Running in a VPC to Custom IAM Policies

The `SageMakerFullAccess` managed policy includes the permissions that you need to use models configured for Amazon VPC access with an endpoint. These permissions allow SageMaker to create an elastic network interface and attach it to model containers running in a VPC. If you use your own IAM policy, you must add the following permissions to that policy to use models configured for VPC access.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:DescribeNetworkInterfaces",  
                "ec2:CreateNetworkInterface",  
                "ec2:AssociateNetworkInterface",  
                "ec2:DeleteNetworkInterface"  
            ],  
            "Resource": "  
                arn:aws:sagemaker:  
                  
                <region>:  
                <account>:network-interface/  
                <interfaceId>  
            "  
        }  
    ]  
}
```

```
        "Action": [
            "ec2:DescribeVpcEndpoints",
            "ec2:DescribeDhcpOptions",
            "ec2:DescribeVpcs",
            "ec2:DescribeSubnets",
            "ec2:DescribeSecurityGroups",
            "ec2:DescribeNetworkInterfaces",
            "ec2:DeleteNetworkInterfacePermission",
            "ec2:DeleteNetworkInterface",
            "ec2>CreateNetworkInterfacePermission",
            "ec2>CreateNetworkInterface"
        ],
        "Resource": "*"
    }
]
```

For more information about the `SageMakerFullAccess` managed policy, see [AmazonSageMakerFullAccess \(p. 2465\)](#).

## Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, `http://s3-aws-region.amazonaws.com/MyBucket`) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your models resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see [Routing for Gateway Endpoints](#) in the *Amazon VPC User Guide*.

## Connect to Resources Outside Your VPC

If you configure your VPC so that it doesn't have internet access, models that use that VPC do not have access to resources outside your VPC. If your model needs access to resources outside your VPC, provide access with one of the following options:

- If your model needs access to an Amazon service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see [VPC Endpoints](#) in the *Amazon VPC User Guide*. For information about creating an interface VPC endpoint, see [Interface VPC Endpoints \(Amazon PrivateLink\)](#) in the *Amazon VPC User Guide*.
- If your model needs access to an Amazon service that doesn't support interface VPC endpoints or to a resource outside of Amazon, create a NAT gateway and configure your security groups to allow outbound connections. For information about setting up a NAT gateway for your VPC, see [Scenario 2: VPC with Public and Private Subnets \(NAT\)](#) in the *Amazon Virtual Private Cloud User Guide*.

## Give SageMaker Training Jobs Access to Resources in Your Amazon VPC

SageMaker runs training jobs in an Amazon Virtual Private Cloud by default. However, training containers access Amazon resources—such as the Amazon Simple Storage Service (Amazon S3) buckets where you store training data and model artifacts—over the internet.

To control access to your data and training containers, we recommend that you create a private VPC and configure it so that they aren't accessible over the internet. For information about creating and configuring a VPC, see [Getting Started With Amazon VPC](#) in the *Amazon VPC User Guide*. Using a VPC helps to protect your training containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your training containers by using VPC flow logs. For more information, see [VPC Flow Logs](#) in the *Amazon VPC User Guide*.

You specify your private VPC configuration when you create training jobs by specifying subnets and security groups. When you specify the subnets and security groups, SageMaker creates *elastic network interfaces* that are associated with your security groups in one of the subnets. Network interfaces allow your training containers to connect to resources in your VPC. For information about network interfaces, see [Elastic Network Interfaces](#) in the *Amazon VPC User Guide*.

**Note**

For training jobs, you can configure only subnets with a default tenancy VPC in which your instance runs on shared hardware. For more information on the tenancy attribute for VPCs, see [Dedicated Instances](#).

## Configure a Training Job for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the `VpcConfig` request parameter of the [CreateTrainingJob](#) API, or provide this information when you create a training job in the SageMaker console. SageMaker uses this information to create network interfaces and attach them to your training containers. The network interfaces provide your training containers with a network connection within your VPC that is not connected to the internet. They also enable your training job to connect to resources in your private VPC.

The following is an example of the `VpcConfig` parameter that you include in your call to `CreateTrainingJob`:

```
VpcConfig: {  
    "Subnets": [  
        "subnet-0123456789abcdef0",  
        "subnet-0123456789abcdef1",  
        "subnet-0123456789abcdef2"  
    ],  
    "SecurityGroupIds": [  
        "sg-0123456789abcdef0"  
    ]  
}
```

## Configure Your Private VPC for SageMaker Training

When configuring the private VPC for your SageMaker training jobs, use the following guidelines. For information about setting up a VPC, see [Working with VPCs and Subnets](#) in the *Amazon VPC User Guide*.

### Topics

- [Ensure That Subnets Have Enough IP Addresses \(p. 2514\)](#)
- [Create an Amazon S3 VPC Endpoint \(p. 2514\)](#)
- [Use a Custom Endpoint Policy to Restrict Access to S3 \(p. 2515\)](#)
- [Configure Route Tables \(p. 2516\)](#)
- [Configure the VPC Security Group \(p. 2516\)](#)
- [Connect to Resources Outside Your VPC \(p. 2516\)](#)

### Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each instance in a training job. For more information, see [VPC and Subnet Sizing for IPv4](#) in the *Amazon VPC User Guide*.

### Create an Amazon S3 VPC Endpoint

If you configure your VPC so that training containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your training data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your training containers to access the buckets where you

store your data and model artifacts . We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see [Endpoints for Amazon S3](#).

#### To create an S3 VPC endpoint:

1. Open the Amazon VPC console at <https://console.amazonaws.cn/vpc/>.
2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**
3. For **Service Name**, search for **com.amazonaws.region.s3**, where **region** is the name of the region where your VPC resides.
4. Choose the **Gateway** type.
5. For **VPC**, choose the VPC you want to use for this endpoint.
6. For **Configure route tables**, select the route tables to be used by the endpoint. The VPC service automatically adds a route to each route table you select that points any S3 traffic to the new endpoint.
7. For **Policy**, choose **Full Access** to allow full access to the S3 service by any user or service within the VPC. Choose **Custom** to restrict access further. For information, see [Use a Custom Endpoint Policy to Restrict Access to S3 \(p. 2515\)](#).

### Use a Custom Endpoint Policy to Restrict Access to S3

The default endpoint policy allows full access to S3 for any user or service in your VPC. To further restrict access to S3, create a custom endpoint policy. For more information, see [Using Endpoint Policies for Amazon S3](#). You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see [Using Amazon S3 Bucket Policies](#).

#### Restrict Package Installation on the Training Container

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the training container. If you don't want users to install packages from that repository, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```
{  
    "Statement": [  
        {  
            "Sid": "AmazonLinuxAMIRRepositoryAccess",  
            "Principal": "*",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Effect": "Deny",  
            "Resource": [  
                "arn:aws:s3:::packages.*.amazonaws.com/*",  
                "arn:aws:s3:::repo.*.amazonaws.com/*"  
            ]  
        }  
    ]  
}  
  
{  
    "Statement": [  
        { "Sid": "AmazonLinux2AMIRRepositoryAccess",  
          "Principal": "*",  
          "Action": [  
              "s3:GetObject"  
            ],  
          "Effect": "Deny",  
          "Resource": [  
        ]  
    ]  
}
```

```
        "arn:aws:s3:::amazonlinux.*.amazonaws.com/*"
    ]
}
}
```

## Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, `http://s3-aws-region.amazonaws.com/MyBucket`) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your training jobs resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see [Routing for Gateway Endpoints](#) in the *Amazon VPC User Guide*.

## Configure the VPC Security Group

In distributed training, you must allow communication between the different containers in the same training job. To do that, configure a rule for your security group that allows inbound connections between members of the same security group. For information, see [Security Group Rules](#).

## Connect to Resources Outside Your VPC

If you configure your VPC so that it doesn't have internet access, training jobs that use that VPC do not have access to resources outside your VPC. If your training job needs access to resources outside your VPC, provide access with one of the following options:

- If your training job needs access to an Amazon service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see [VPC Endpoints](#) in the *Amazon VPC User Guide*. For information about creating an interface VPC endpoint, see [Interface VPC Endpoints \(Amazon PrivateLink\)](#) in the *Amazon VPC User Guide*.
- If your training job needs access to an Amazon service that doesn't support interface VPC endpoints or to a resource outside of Amazon, create a NAT gateway and configure your security groups to allow outbound connections. For information about setting up a NAT gateway for your VPC, see [Scenario 2: VPC with Public and Private Subnets \(NAT\)](#) in the *Amazon Virtual Private Cloud User Guide*.

# Give Batch Transform Jobs Access to Resources in Your Amazon VPC

SageMaker runs batch transform jobs in an Amazon Virtual Private Cloud by default. However, model containers access Amazon resources—such as the Amazon S3 buckets where you store your data and model artifacts—over the internet.

To control access to your model containers and data, we recommend that you create a private VPC and configure it so that they aren't accessible over the internet. For information about creating and configuring a VPC, see [Getting Started With Amazon VPC](#) in the *Amazon VPC User Guide*. Using a VPC helps to protect your model containers and data because you can configure your VPC so that it is not connected to the internet. Using a VPC also allows you to monitor all network traffic in and out of your model containers by using VPC flow logs. For more information, see [VPC Flow Logs](#) in the *Amazon VPC User Guide*.

You specify your private VPC configuration when you create a model by specifying subnets and security groups. You then specify the same model when you create a batch transform job. When you specify the subnets and security groups, SageMaker creates *elastic network interfaces* that are associated with your security groups in one of the subnets. Network interfaces allow your model containers to connect to resources in your VPC. For information about network interfaces, see [Elastic Network Interfaces](#) in the *Amazon VPC User Guide*.

## Configure a Batch Transform Job for Amazon VPC Access

To specify subnets and security groups in your private VPC, use the `VpcConfig` request parameter of the [CreateModel](#) API, or provide this information when you create a model in the SageMaker console. Then specify the same model in the `ModelName` request parameter of the [CreateTransformJob](#) API, or in the **Model name** field when you create a transform job in the SageMaker console. SageMaker uses this information to create network interfaces and attach them to your model containers. The network interfaces provide your model containers with a network connection within your VPC that is not connected to the internet. They also enable your transform job to connect to resources in your private VPC.

The following is an example of the `VpcConfig` parameter that you include in your call to [CreateModel](#):

```
VpcConfig: {  
    "Subnets": [  
        "subnet-0123456789abcdef0",  
        "subnet-0123456789abcdef1",  
        "subnet-0123456789abcdef2"  
    ],  
    "SecurityGroupIds": [  
        "sg-0123456789abcdef0"  
    ]  
}
```

If you are creating a model using the [CreateModel](#) API operation, the IAM execution role that you use to create your model must include the permissions described in [CreateModel API: Execution Role Permissions \(p. 2462\)](#), including the following permissions required for a private VPC.

When creating a model in the console, if you select **Create a new role** in the **Model Settings** section, the [AmazonSageMakerFullAccess](#) policy used to create the role will already contain these permissions. If you select **Enter a custom IAM role ARN** or **Use existing role**, the role ARN that you specify must have an execution policy attached with the following permissions.

```
{  
    "Effect": "Allow",  
    "Action": [  
        "ec2:CreateNetworkInterface",  
        "ec2:CreateNetworkInterfacePermission",  
        "ec2:DeleteNetworkInterface",  
        "ec2:DeleteNetworkInterfacePermission",  
        "ec2:DescribeNetworkInterfaces",  
        "ec2:DescribeVpcs",  
        "ec2:DescribeDhcpOptions",  
        "ec2:DescribeSubnets",  
        "ec2:DescribeSecurityGroups"
```

## Configure Your Private VPC for SageMaker Batch Transform

When configuring the private VPC for your SageMaker batch transform jobs, use the following guidelines. For information about setting up a VPC, see [Working with VPCs and Subnets](#) in the *Amazon VPC User Guide*.

### Topics

- [Ensure That Subnets Have Enough IP Addresses \(p. 2518\)](#)
- [Create an Amazon S3 VPC Endpoint \(p. 2518\)](#)
- [Use a Custom Endpoint Policy to Restrict Access to S3 \(p. 2518\)](#)
- [Configure Route Tables \(p. 2519\)](#)

- [Configure the VPC Security Group \(p. 2519\)](#)
- [Connect to Resources Outside Your VPC \(p. 2519\)](#)

## Ensure That Subnets Have Enough IP Addresses

Your VPC subnets should have at least two private IP addresses for each instance in a transform job. For more information, see [VPC and Subnet Sizing for IPv4](#) in the *Amazon VPC User Guide*.

## Create an Amazon S3 VPC Endpoint

If you configure your VPC so that model containers don't have access to the internet, they can't connect to the Amazon S3 buckets that contain your data unless you create a VPC endpoint that allows access. By creating a VPC endpoint, you allow your model containers to access the buckets where you store your data and model artifacts . We recommend that you also create a custom policy that allows only requests from your private VPC to access to your S3 buckets. For more information, see [Endpoints for Amazon S3](#).

### To create an S3 VPC endpoint:

1. Open the Amazon VPC console at <https://console.amazonaws.cn/vpc/>.
2. In the navigation pane, choose **Endpoints**, then choose **Create Endpoint**
3. For **Service Name**, choose **com.amazonaws.region.s3**, where **region** is the name of the region where your VPC resides.
4. For **VPC**, choose the VPC you want to use for this endpoint.
5. For **Configure route tables**, select the route tables to be used by the endpoint. The VPC service automatically adds a route to each route table you select that points any S3 traffic to the new endpoint.
6. For **Policy**, choose **Full Access** to allow full access to the S3 service by any user or service within the VPC. Choose **Custom** to restrict access further. For information, see [Use a Custom Endpoint Policy to Restrict Access to S3 \(p. 2518\)](#).

## Use a Custom Endpoint Policy to Restrict Access to S3

The default endpoint policy allows full access to S3 for any user or service in your VPC. To further restrict access to S3, create a custom endpoint policy. For more information, see [Using Endpoint Policies for Amazon S3](#). You can also use a bucket policy to restrict access to your S3 buckets to only traffic that comes from your Amazon VPC. For information, see [Using Amazon S3 Bucket Policies](#).

### Restrict Package Installation on the Model Container

The default endpoint policy allows users to install packages from the Amazon Linux and Amazon Linux 2 repositories on the training container. If you don't want users to install packages from that repository, create a custom endpoint policy that explicitly denies access to the Amazon Linux and Amazon Linux 2 repositories. The following is an example of a policy that denies access to these repositories:

```
{  
    "Statement": [  
        {  
            "Sid": "AmazonLinuxAMIRRepositoryAccess",  
            "Principal": "*",  
            "Action": [  
                "s3:GetObject"  
            ],  
            "Effect": "Deny",  
            "Resource": [  
                "arn:aws:s3:::packages.*.amazonaws.com/*",  
                "arn:aws:s3:::packages.*.amazonaws.com/*/  
            ]  
        }  
    ]  
}
```

```
        "arn:aws:s3:::repo.*.amazonaws.com/*"
    }
}
{
  "Statement": [
    { "Sid": "AmazonLinux2AMIRRepositoryAccess",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Deny",
      "Resource": [
        "arn:aws:s3:::amazonlinux.*.amazonaws.com/*"
      ]
    }
  ]
}
```

## Configure Route Tables

Use default DNS settings for your endpoint route table, so that standard Amazon S3 URLs (for example, `http://s3-aws-region.amazonaws.com/MyBucket`) resolve. If you don't use default DNS settings, ensure that the URLs that you use to specify the locations of the data in your batch transform jobs resolve by configuring the endpoint route tables. For information about VPC endpoint route tables, see [Routing for Gateway Endpoints in the Amazon VPC User Guide](#).

## Configure the VPC Security Group

In distributed batch transform, you must allow communication between the different containers in the same batch transform job. To do that, configure a rule for your security group that allows inbound connections between members of the same security group. For information, see [Security Group Rules](#).

## Connect to Resources Outside Your VPC

If you configure your VPC so that it doesn't have internet access, batch transform jobs that use that VPC do not have access to resources outside your VPC. If your batch transform job needs access to resources outside your VPC, provide access with one of the following options:

- If your batch transform job needs access to an Amazon service that supports interface VPC endpoints, create an endpoint to connect to that service. For a list of services that support interface endpoints, see [VPC Endpoints](#) in the *Amazon VPC User Guide*. For information about creating an interface VPC endpoint, see [Interface VPC Endpoints \(Amazon PrivateLink\)](#) in the *Amazon VPC User Guide*.
- If your batch transform job needs access to an Amazon service that doesn't support interface VPC endpoints or to a resource outside of Amazon, create a NAT gateway and configure your security groups to allow outbound connections. For information about setting up a NAT gateway for your VPC, see [Scenario 2: VPC with Public and Private Subnets \(NAT\)](#) in the *Amazon Virtual Private Cloud User Guide*.

## Give Amazon SageMaker Clarify Jobs Access to Resources in Your Amazon VPC

An Amazon SageMaker Clarify job is a SageMaker processing job. It is deployed in an Amazon Virtual Private Cloud by default and so needs to have access over the public internet to Amazon resources such as the Amazon S3 buckets where you store data.

To control access to your data and SageMaker Clarify jobs, we recommend that you create a private Amazon VPC and configure it so that your jobs aren't accessible over the public internet. For information about creating and configuring an Amazon VPC for processing jobs, see [Give SageMaker Processing Jobs Access to Resources in Your Amazon VPC](#).

This document explains how to add additional Amazon VPC configurations that meet the pre-requirements for SageMaker Clarify jobs.

### Topics

- [Configure a SageMaker Clarify Job for Amazon VPC Access \(p. 2520\)](#)
- [Configure Your Private Amazon VPC for SageMaker Clarify jobs \(p. 2522\)](#)

## Configure a SageMaker Clarify Job for Amazon VPC Access

You need to specify subnets and security groups when configuring your private Amazon VPC for SageMaker Clarify jobs and to enable the job to get inferences from the SageMaker model when computing post-training bias metrics and feature contributions that help explain model predictions.

### Topics

- [SageMaker Clarify Job Amazon VPC Subnets and Security Groups \(p. 2520\)](#)
- [Configure a Model Amazon VPC for Inference \(p. 2521\)](#)

## SageMaker Clarify Job Amazon VPC Subnets and Security Groups

Subnets and security groups in your private Amazon VPC can be assigned to a SageMaker Clarify job in various ways, depending on how you create the job.

- **SageMaker console:** Provide this information when you create the job in the [SageMaker Dashboard](#). From the **Processing** menu, choose **Processing jobs**, then choose **Create processing job**. Select the **VPC** option in the **Network** panel and provide the subnets and security groups using the dropdown lists. Make sure network isolation option provided in this panel is set to disabled.
- **SageMaker API:** Use the `NetworkConfig.VpcConfig` request parameter of the [CreateProcessingJob](#) API, as shown in the following example:

```
"NetworkConfig": {  
    "VpcConfig": {  
        "Subnets": [  
            "subnet-0123456789abcdef0",  
            "subnet-0123456789abcdef1",  
            "subnet-0123456789abcdef2"  
        ],  
        "SecurityGroupIds": [  
            "sg-0123456789abcdef0"  
        ]  
    }  
}
```

- **SageMaker Python SDK:** Use the `NetworkConfig` parameter of the [SageMakerClarifyProcessor](#) API or [Processor](#) API, as shown in the following example:

```
from sagemaker.network import NetworkConfig  
network_config = NetworkConfig(  
    subnets=[  
        "subnet-0123456789abcdef0",  
        "subnet-0123456789abcdef1",  
        "subnet-0123456789abcdef2",
```

```
],
  security_group_ids=[
    "sg-0123456789abcdef0",
  ],
)
```

SageMaker uses the information to create network interfaces and attach them to the SageMaker Clarify job. The network interfaces provide a SageMaker Clarify job with a network connection within your Amazon VPC that is not connected to the public internet. They also enable the SageMaker Clarify job to connect to resources in your private Amazon VPC.

## Configure a Model Amazon VPC for Inference

In order to compute post-training bias metrics and explainability, the SageMaker Clarify job needs to get inferences from the SageMaker model that is specified by the `model_name` parameter of the [analysis configuration](#) for the SageMaker Clarify processing job. Alternatively, if you use the `SageMakerClarifyProcessor` API in the SageMaker Python SDK, the job needs to get the `model_name` specified by the [ModelConfig](#) class. To accomplish this, the SageMaker Clarify job creates an ephemeral endpoint with the model, known as a *shadow endpoint*, and then applies the Amazon VPC configuration of the model to the shadow endpoint.

### Note

The network isolation option of both the SageMaker Clarify job and the model must be disabled (by default the option is disabled) so that the SageMaker Clarify job can communicate with the shadow endpoint.

To specify subnets and security groups in your private Amazon VPC to the SageMaker model, use the `VpcConfig` request parameter of the [CreateModel](#) API or provide this information when you create the model using the SageMaker dashboard in the console. The following is an example of the `vpcConfig` parameter that you include in your call to `CreateModel`:

```
"VpcConfig": {
  "Subnets": [
    "subnet-0123456789abcdef0",
    "subnet-0123456789abcdef1",
    "subnet-0123456789abcdef2"
  ],
  "SecurityGroupIds": [
    "sg-0123456789abcdef0"
  ]
}
```

You can specify the number of instances of the shadow endpoint to launch with the `initial_instance_count` parameter of the [analysis configuration](#) for the SageMaker Clarify processing job. Alternatively, if you use the `SageMakerClarifyProcessor` API in the SageMaker Python SDK, the job needs to get the `instance_count` specified by the [ModelConfig](#) class.

### Note

Even if you only request one instance when creating the shadow endpoint, you need at least two subnets in the model's [ModelConfig](#) in distinct availability zones. Otherwise the shadow endpoint creation fails with the following error:  
`ClientError: Error hosting endpoint sagemaker-clarify-endpoint-XXX: Failed. Reason: Unable to locate at least 2 availability zone(s) with the requested instance type YYY that overlap with SageMaker subnets.`

If your model requires model files in Amazon S3, then the model Amazon VPC needs to have an Amazon S3 VPC endpoint. For more information about creating and configuring an Amazon VPC for SageMaker models, see [Give SageMaker Hosted Endpoints Access to Resources in Your Amazon VPC \(p. 2510\)](#).

## Configure Your Private Amazon VPC for SageMaker Clarify jobs

In general, you can follow the steps in [Configure Your Private VPC for SageMaker Processing](#) to configure your private Amazon VPC for SageMaker Clarify jobs. Here are some highlights and special requirements for SageMaker Clarify jobs.

### Topics

- [Connect to Resources Outside Your Amazon VPC \(p. 2522\)](#)
- [Configure the Amazon VPC Security Group \(p. 2522\)](#)

### Connect to Resources Outside Your Amazon VPC

If you configure your Amazon VPC so that it does not have public internet access, then some additional setup is required to grant SageMaker Clarify jobs access to resources and services outside of your Amazon VPC. For example, an Amazon S3 VPC Endpoint is required because a SageMaker Clarify job needs to load a dataset from an S3 bucket as well as save the analysis results to an S3 bucket. For more information, see [Create an Amazon S3 VPC Endpoint](#) for the creation guide. In addition, if a SageMaker Clarify job needs to get inferences from the shadow endpoint, then it needs to call several more Amazon services.

- **Create an Amazon SageMaker API service VPC endpoint:** The SageMaker Clarify job needs to call the Amazon SageMaker API service to manipulate the shadow endpoint, or describe a SageMaker model for Amazon VPC validation. You can follow the guidance provided in the [Securing all Amazon SageMaker API calls with Amazon PrivateLink](#) blog to create an Amazon SageMaker API VPC endpoint that allows the SageMaker Clarify job to make the service calls. Note that the service name of Amazon SageMaker API service is `com.amazonaws.region.sagemaker.api`, where `region` is the name of the Region where your Amazon VPC resides.
- **Create an Amazon SageMaker Runtime VPC Endpoint:** The SageMaker Clarify job needs to call the Amazon SageMaker runtime service, which routes the invocations to the shadow endpoint. The setup steps are similar to those for the Amazon SageMaker API service. Note that the service name of Amazon SageMaker Runtime service is `com.amazonaws.region.sagemaker.runtime`, where `region` is the name of the Region where your Amazon VPC resides.

### Configure the Amazon VPC Security Group

SageMaker Clarify jobs support distributed processing when two or more processing instances are specified in one of the following ways:

- **SageMaker console:** The `Instance count` is specified in the **Resource configuration** part of the **Job settings** panel on the [Create processing job](#) page.
- **SageMaker API:** The `InstanceCount` is specified when you create the job with the `CreateProcessingJob` API.
- **SageMaker Python SDK:** The `instance_count` is specified when using the `SageMakerClarifyProcessor` API or the `Processor` API.

In distributed processing, you must allow communication between the different instances in the same processing job. To do that, configure a rule for your security group that allows inbound connections between members of the same security group. For information, see [Security group rules](#).

# Monitor Amazon SageMaker

Monitoring is an important part of maintaining the reliability, availability, and performance of SageMaker and your other Amazon solutions. Amazon provides the following monitoring tools to watch SageMaker, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors your Amazon resources and the applications that you run on Amazon in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the [Amazon CloudWatch User Guide](#).
- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from EC2 instances, Amazon CloudTrail, and other sources. CloudWatch Logs can monitor information in the log files and notify you when certain thresholds are met. You can also archive your log data in highly durable storage. For more information, see the [Amazon CloudWatch Logs User Guide](#).
- *Amazon CloudTrail* captures API calls and related events made by or on behalf of your Amazon account and delivers the log files to an Amazon S3 bucket that you specify. You can identify which users and accounts called Amazon, the source IP address from which the calls were made, and when the calls occurred. For more information, see the [Amazon CloudTrail User Guide](#).
- *CloudWatch Events* delivers a near real-time stream of system events that describe changes in Amazon resources. Create CloudWatch Events rules react to a status change in a SageMaker training, hyperparameter tuning, or batch transform job

## Topics

- [Monitor Amazon SageMaker with Amazon CloudWatch \(p. 2523\)](#)
- [Log Amazon SageMaker Events with Amazon CloudWatch \(p. 2534\)](#)
- [Log Amazon SageMaker API Calls with Amazon CloudTrail \(p. 2535\)](#)
- [Automating Amazon SageMaker with Amazon EventBridge \(p. 2537\)](#)

## Monitor Amazon SageMaker with Amazon CloudWatch

You can monitor Amazon SageMaker using Amazon CloudWatch, which collects raw data and processes it into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. However, the Amazon CloudWatch console limits the search to metrics that were updated in the last 2 weeks. This limitation ensures that the most current jobs are shown in your namespace. To graph metrics without using a search, specify its exact name in the source view. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the [Amazon CloudWatch User Guide](#).

### SageMaker Metrics and Dimensions

- [SageMaker Endpoint Invocation Metrics \(p. 2524\)](#)
- [SageMaker Multi-Model Endpoint Metrics \(p. 2525\)](#)
- [SageMaker Jobs and Endpoint Metrics \(p. 2527\)](#)
- [SageMaker Ground Truth Metrics \(p. 2529\)](#)
- [SageMaker Feature Store Metrics \(p. 2531\)](#)

- [SageMaker Pipelines Metrics \(p. 2532\)](#)

## SageMaker Endpoint Invocation Metrics

The AWS/SageMaker namespace includes the following request metrics from calls to [InvokeEndpoint](#).

Metrics are available at a 1-minute frequency.

For information about how long CloudWatch metrics are retained for, see [GetMetricStatistics](#) in the [Amazon CloudWatch API Reference](#).

### Endpoint Invocation Metrics

Metric	Description
<code>Invocation4XXErrors</code>	The number of <code>InvokeEndpoint</code> requests where the model returned a 4xx HTTP response code. For each 4xx response, 1 is sent; otherwise, 0 is sent.  Units: None  Valid statistics: Average, Sum
<code>Invocation5XXErrors</code>	The number of <code>InvokeEndpoint</code> requests where the model returned a 5xx HTTP response code. For each 5xx response, 1 is sent; otherwise, 0 is sent.  Units: None  Valid statistics: Average, Sum
<code>Invocations</code>	The number of <code>InvokeEndpoint</code> requests sent to a model endpoint.  To get the total number of requests sent to a model endpoint, use the Sum statistic.  Units: None  Valid statistics: Sum
<code>InvocationsPerInstance</code>	The number of invocations sent to a model, normalized by <code>InstanceCount</code> in each <code>ProductionVariant</code> . $1/\text{numberOfInstances}$ is sent as the value on each request, where <code>numberOfInstances</code> is the number of active instances for the <code>ProductionVariant</code> behind the endpoint at the time of the request.  Units: None  Valid statistics: Sum
<code>ModelLatency</code>	The interval of time taken by a model to respond as viewed from SageMaker. This interval includes the local communication times taken to send the request and to fetch the response from the container of a model and the time taken to complete the inference in the container.  Units: Microseconds  Valid statistics: Average, Sum, Min, Max, Sample Count
<code>OverheadLatency</code>	The interval of time added to the time taken to respond to a client request by SageMaker overheads. This interval is measured from the time SageMaker receives the request until it returns a response to the client, minus the

Metric	Description
	<p><b>ModelLatency.</b> Overhead latency can vary depending on multiple factors, including request and response payload sizes, request frequency, and authentication/authorization of the request.</p> <p>Units: Microseconds</p> <p>Valid statistics: Average, Sum, Min, Max, Sample Count</p>

#### Dimensions for Endpoint Invocation Metrics

Dimension	Description
EndpointName, VariantName	Filters endpoint invocation metrics for a <code>ProductionVariant</code> of the specified endpoint and variant.

## SageMaker Multi-Model Endpoint Metrics

The `AWS/SageMaker` namespace includes the following model loading metrics from calls to `InvokeEndpoint`.

Metrics are available at a 1-minute frequency.

For information about how long CloudWatch metrics are retained for, see [GetMetricStatistics](#) in the [Amazon CloudWatch API Reference](#).

#### Multi-Model Endpoint Model Loading Metrics

Metric	Description
<code>ModelLoadingWaitTime</code>	<p>The interval of time that an invocation request has waited for the target model to be downloaded, or loaded, or both in order to perform inference.</p> <p>Units: Microseconds</p> <p>Valid statistics: Average, Sum, Min, Max, Sample Count</p>
<code>ModelUnloadingTime</code>	<p>The interval of time that it took to unload the model through the container's <code>UnloadModel</code> API call.</p> <p>Units: Microseconds</p> <p>Valid statistics: Average, Sum, Min, Max, Sample Count</p>
<code>ModelDownloadingTime</code>	<p>The interval of time that it took to download the model from Amazon Simple Storage Service (Amazon S3).</p> <p>Units: Microseconds</p> <p>Valid statistics: Average, Sum, Min, Max, Sample Count</p>
<code>ModelLoadingTime</code>	<p>The interval of time that it took to load the model through the container's <code>LoadModel</code> API call.</p> <p>Units: Microseconds</p>

Metric	Description
	Valid statistics: Average, Sum, Min, Max, Sample Count
ModelCacheHit	<p>The number of <code>InvokeEndpoint</code> requests sent to the multi-model endpoint for which the model was already loaded.</p> <p>The Average statistic shows the ratio of requests for which the model was already loaded.</p> <p>Units: None</p> <p>Valid statistics: Average, Sum, Sample Count</p>

#### Dimensions for Multi-Model Endpoint Model Loading Metrics

Dimension	Description
EndpointName, VariantName	Filters endpoint invocation metrics for a <code>ProductionVariant</code> of the specified endpoint and variant.

The `/aws/sagemaker/Endpoints` namespaces include the following instance metrics from calls to `InvokeEndpoint`.

Metrics are available at a 1-minute frequency.

For information about how long CloudWatch metrics are retained for, see [GetMetricStatistics](#) in the [Amazon CloudWatch API Reference](#).

#### Multi-Model Endpoint Model Instance Metrics

Metric	Description
LoadedModelCount	<p>The number of models loaded in the containers of the multi-model endpoint. This metric is emitted per instance.</p> <p>The Average statistic with a period of 1 minute tells you the average number of models loaded per instance.</p> <p>The Sum statistic tells you the total number of models loaded across all instances in the endpoint.</p> <p>The models that this metric tracks are not necessarily unique because a model might be loaded in multiple containers at the endpoint.</p> <p>Units: None</p> <p>Valid statistics: Average, Sum, Min, Max, Sample Count</p>

#### Dimensions for Multi-Model Endpoint Model Loading Metrics

Dimension	Description
EndpointName, VariantName	Filters endpoint invocation metrics for a <code>ProductionVariant</code> of the specified endpoint and variant.

## SageMaker Jobs and Endpoint Metrics

The `/aws/sagemaker/ProcessingJobs`, `/aws/sagemaker/TrainingJobs`, `/aws/sagemaker/TransformJobs` and `/aws/sagemaker/Endpoints` namespaces include the following metrics for the training jobs and endpoint instances.

Metrics are available at a 1-minute frequency.

### Processing Job, Training Job, Batch Transform Job, and Endpoint Instance Metrics

Metric	Description
CPUUtilization	<p>The sum of each individual CPU core's utilization. The CPU utilization of each core can range between 0 and 100. For example, if there are four CPUs, <code>CPUUtilization</code> can range from 0% to 400%.</p> <p>For processing jobs, the value is the CPU utilization of the processing container on the instance.</p> <p>For training jobs, the value is the CPU utilization of the algorithm container on the instance.</p> <p>For batch transform jobs, the value is the CPU utilization of the transform container on the instance.</p> <p>For endpoint variants, the value is the sum of the CPU utilization of the primary and supplementary containers on the instance.</p> <p><b>Note</b> For multi-instance, each instance reports CPU utilization metrics. However, the default view in CloudWatch shows the average CPU utilization across all instances.</p> <p>Units: Percent</p>
MemoryUtilization	<p>The percentage of memory that is used by the containers on an instance. This value can range between 0% and 100%.</p> <p>For processing jobs, the value is the memory utilization of the processing container on the instance.</p> <p>For training jobs, the value is the memory utilization of the algorithm container on the instance.</p> <p>For batch transform jobs, the value is the memory utilization of the transform container on the instance.</p> <p>For endpoint variants, the value is the sum of the memory utilization of the primary and supplementary containers on the instance.</p> <p>Units: Percent</p> <p><b>Note</b> For multi-instance, each instance reports memory utilization metrics. However, the default view in CloudWatch shows the average memory utilization across all instances.</p>
GPUUtilization	<p>The percentage of GPU units that are used by the containers on an instance. The value can range between 0 and 100 and is multiplied by the number of GPUs. For example, if there are four GPUs, <code>GPUUtilization</code> can range from 0% to 400%.</p>

Metric	Description
	<p>For processing jobs, the value is the GPU utilization of the processing container on the instance.</p> <p>For training jobs, the value is the GPU utilization of the algorithm container on the instance.</p> <p>For batch transform jobs, the value is the GPU utilization of the transform container on the instance.</p> <p>For endpoint variants, the value is the sum of the GPU utilization of the primary and supplementary containers on the instance.</p> <p><b>Note</b> For multi-instance, each instance reports GPU utilization metrics. However, the default view in CloudWatch shows the average GPU utilization across all instances.</p> <p>Units: Percent</p>
GPUMemoryUtilization	<p>The percentage of GPU memory used by the containers on an instance. The value can range between 0 and 100 and is multiplied by the number of GPUs. For example, if there are four GPUs, GPUMemoryUtilization can range from 0% to 400%.</p> <p>For processing jobs, the value is the GPU memory utilization of the processing container on the instance.</p> <p>For training jobs, the value is the GPU memory utilization of the algorithm container on the instance.</p> <p>For batch transform jobs, the value is the GPU memory utilization of the transform container on the instance.</p> <p>For endpoint variants, the value is the sum of the GPU memory utilization of the primary and supplementary containers on the instance.</p> <p><b>Note</b> For multi-instance, each instance reports GPU memory utilization metrics. However, the default view in CloudWatch shows the average GPU memory utilization across all instances.</p> <p>Units: Percent</p>

Metric	Description
DiskUtilization	<p>The percentage of disk space used by the containers on an instance uses. This value can range between 0% and 100%. This metric is not supported for batch transform jobs.</p> <p>For processing jobs, the value is the disk space utilization of the processing container on the instance.</p> <p>For training jobs, the value is the disk space utilization of the algorithm container on the instance.</p> <p>For endpoint variants, the value is the sum of the disk space utilization of the primary and supplementary containers on the instance.</p> <p>Units: Percent</p> <p><b>Note</b></p> <p>For multi-instance, each instance reports disk utilization metrics. However, the default view in CloudWatch shows the average disk utilization across all instances.</p>

#### Dimensions for Processing Job, Training Job and Batch Transform Job Instance Metrics

Dimension	Description
Host	<p>For processing jobs, the value for this dimension has the format [processing-job-name]/algo-[instance-number-in-cluster]. Use this dimension to filter instance metrics for the specified processing job and instance. This dimension format is present only in the /aws/sagemaker/ProcessingJobs namespace.</p> <p>For training jobs, the value for this dimension has the format [training-job-name]/algo-[instance-number-in-cluster]. Use this dimension to filter instance metrics for the specified training job and instance. This dimension format is present only in the /aws/sagemaker/TrainingJobs namespace.</p> <p>For batch transform jobs, the value for this dimension has the format [transform-job-name]/[instance-id]. Use this dimension to filter instance metrics for the specified batch transform job and instance. This dimension format is present only in the /aws/sagemaker/TransformJobs namespace.</p>

## SageMaker Ground Truth Metrics

### Ground Truth Metrics

Metric	Description
ActiveWorkers	A single active worker on a private work team submitted, released, or declined a task. To get the total number of active workers, use the Sum statistic. Ground Truth attempts to deliver each individual ActiveWorkers event once. If this delivery is unsuccessful, this metric may not report the total number of active workers

Metric	Description
	Units: None Valid statistics: Sum, Sample Count
DatasetObjectsAutoAnnotated	<b>The number</b> of dataset objects auto-annotated in a labeling job. This metric is only emitted when automated labeling is enabled. To view the labeling job progress, use the Max metric. Units: None Valid statistics: Max
DatasetObjectsHumanAnnotated	<b>The number</b> of dataset objects annotated by a human in a labeling job. To view the labeling job progress, use the Max metric. Units: None Valid statistics: Max
DatasetObjectsLabelingFailed	<b>The number</b> of dataset objects that failed labeling in a labeling job. To view the labeling job progress, use the Max metric. Units: None Valid statistics: Max
JobsFailed	A single labeling job failed. To get the total number of labeling jobs that failed, use the Sum statistic. Units: None Valid statistics: Sum, Sample Count
JobsSucceeded	A single labeling job succeeded. To get the total number of labeling jobs that succeeded, use the Sum statistic. Units: None Valid statistics: Sum, Sample Count
JobsStopped	A single labeling job was stopped. To get the total number of labeling jobs that were stopped, use the Sum statistic. Units: None Valid statistics: Sum, Sample Count
TasksAccepted	A single task was accepted by a worker. To get the total number of tasks accepted by workers, use the Sum statistic. Ground Truth attempts to deliver each individual TaskAccepted event once. If this delivery is unsuccessful, this metric may not report the total number of tasks accepted. Units: None Valid statistics: Sum, Sample Count

Metric	Description
TasksDeclined	<p>A single task was declined by a worker. To get the total number of tasks declined by workers, use the Sum statistic. Ground Truth attempts to deliver each individual TasksDeclined event once. If this delivery is unsuccessful, this metric may not report the total number of tasks declined.</p> <p>Units: None</p> <p>Valid Statistics: Sum, Sample Count</p>
TasksReturned	<p>A single task was returned. To get the total number of tasks returned, use the Sum statistic. Ground Truth attempts to deliver each individual TasksReturned event once. If this delivery is unsuccessful, this metric may not report the total number of tasks returned.</p> <p>Units: None</p> <p>Valid statistics: Sum, Sample Count</p>
TasksSubmitted	<p>A single task was submitted/completed by a private worker. To get the total number of tasks submitted by workers, use the Sum statistic. Ground Truth attempts to deliver each individual TasksSubmitted event once. If this delivery is unsuccessful, this metric may not report the total number of tasks submitted.</p> <p>Units: None</p> <p>Valid statistics: Sum, Sample Count</p>
TimeSpent	<p>Time spent on a task completed by a private worker. This metric does not include time when a worker paused or took a break. Ground Truth attempts to deliver each TimeSpent event once. If this delivery is unsuccessful, this metric may not report the total amount of time spent.</p> <p>Units: Seconds</p> <p>Valid statistics: Sum, Sample Count</p>
TotalDatasetObjects	<p>The number of dataset objects labeled successfully in a labeling job. To view the labeling job progress, use the Max metric.</p> <p>Units: None</p> <p>Valid statistics: Max</p>

#### Dimensions for Dataset Object Metrics

Dimension	Description
LabelingJobName	Filters dataset object count metrics for a labeling job.

## SageMaker Feature Store Metrics

### Feature Store Metrics

Metric	Description
ConsumedReadRequests	The number of consumed read units over the specified time period. You can retrieve the consumed read units for a feature store runtime operation and its corresponding feature group.  Units: None  Valid statistics: All
ConsumedWriteRequests	The number of consumed write units over the specified time period. You can retrieve the consumed write units for a feature store runtime operation and its corresponding feature group.  Units: None  Valid statistics: All

#### Dimensions for Feature Store Metrics

Dimension	Description
FeatureGroupName, OperationName	Filters feature store runtime operation metrics of the specified feature group.

## SageMaker Pipelines Metrics

The `AWS/Sagemaker/ModelBuildingPipeline` namespace includes the following metrics for pipeline executions.

Two categories of Pipelines execution metrics are available:

- **Execution Metrics across All Pipelines** – Account level pipeline execution metrics (for all pipelines in the current account)
- **Execution Metrics by Pipeline** – Pipeline execution metrics per pipeline

Metrics are available at a 1-minute frequency.

#### Pipelines Execution Metrics

Metric	Description
ExecutionStarted	The number of pipeline executions that started.  Units: Count  Valid statistics: Average, Sum
ExecutionFailed	The number of pipeline executions that failed.  Units: Count  Valid statistics: Average, Sum

Metric	Description
<code>ExecutionSucceeded</code>	The number of pipeline executions that succeeded.  Units: Count  Valid statistics: Average, Sum
<code>ExecutionStopped</code>	The number of pipeline executions that stopped.  Units: Count  Valid statistics: Average, Sum
<code>ExecutionDuration</code>	The duration in milliseconds that the pipeline execution ran.  Units: Milliseconds  Valid statistics: Average, Sum, Min, Max, Sample Count

#### Dimensions for Execution Metrics by Pipeline

Dimension	Description
<code>PipelineName</code>	Filters pipeline execution metrics for a specified pipeline.

#### Pipelines Step Metrics

The `AWS/Sagemaker/ModelBuildingPipeline` namespace includes the following metrics for pipeline steps.

Metrics are available at a 1-minute frequency.

Metric	Description
<code>StepStarted</code>	The number of steps that started.  Units: Count  Valid statistics: Average, Sum
<code>StepFailed</code>	The number of steps that failed.  Units: Count  Valid statistics: Average, Sum
<code>StepSucceeded</code>	The number of steps that succeeded.  Units: Count  Valid statistics: Average, Sum
<code>StepStopped</code>	The number of steps that stopped.  Units: Count  Valid statistics: Average, Sum

Metric	Description
StepDuration	The duration in milliseconds that the step ran. Units: Milliseconds Valid statistics: Average, Sum, Min, Max, Sample Count

#### Dimensions for Pipelines Step Metrics

Dimension	Description
PipelineName, StepName	Filters step metrics for a specified pipeline and step.

## Log Amazon SageMaker Events with Amazon CloudWatch

To help you debug your processing jobs, training jobs, endpoints, transform jobs, notebook instances, and notebook instance lifecycle configurations, anything an algorithm container, a model container, or a notebook instance lifecycle configuration sends to `stdout` or `stderr` is also sent to Amazon CloudWatch Logs. In addition to debugging, you can use these for progress analysis.

### Logs

The following table lists all of the logs provided by Amazon SageMaker.

### Logs

Log Group Name	Log Stream Name
/aws/sagemaker/ Endpoints/ [EndpointName]	[production-variant-name]/[instance-id]
	[production-variant-name]/[instance-id]/[container-name provided in SageMaker model] (For Inference Pipelines)
/aws/sagemaker/ groundtruth/ WorkerActivity	aws/sagemaker/groundtruth/worker-activity/[requester-AWS-Id]-[region]/[timestamp]
/aws/sagemaker/ LabelingJobs	[labeling-job-name]
/aws/sagemaker/ NotebookInstances	[notebook-instance-name]/[LifecycleConfigHook]
	[notebook-instance-name]/jupyter.log
/aws/sagemaker/ ProcessingJobs	[processing-job-name]/[hostname]-[epoch_timestamp]
/aws/sagemaker/ studio	[domain-id]/[user-profile-name]/[app-type]/[app-name]
/aws/sagemaker/ TrainingJobs	[training-job-name]/algo-[instance-number-in-cluster]-[epoch_timestamp]

Log Group Name	Log Stream Name
/aws/sagemaker/ TransformJobs	[transform-job-name]/[instance-id]-[epoch_timestamp]
	[transform-job-name]/[instance-id]-[epoch_timestamp]/data-log
	[transform-job-name]/[instance-id]-[epoch_timestamp]/ [container-name provided in SageMaker model] (For Inference Pipelines)

**Note**

1. The /aws/sagemaker/NotebookInstances/[LifecycleConfigHook] log stream is created when you create a notebook instance with a lifecycle configuration. For more information, see [Customize a Notebook Instance Using a Lifecycle Configuration Script \(p. 124\)](#).
2. For Inference Pipelines, if you don't provide container names, the platform uses \*\*container-1, container-2\*\*, and so on, corresponding to the order provided in the SageMaker model.

For more information about logging events with CloudWatch logging, see [What is Amazon CloudWatch Logs?](#) in the *Amazon CloudWatch User Guide*.

## Log Amazon SageMaker API Calls with Amazon CloudTrail

Amazon SageMaker is integrated with Amazon CloudTrail, a service that provides a record of actions taken by a user, role, or an Amazon service in SageMaker. CloudTrail captures all API calls for SageMaker, with the exception of [InvokeEndpoint](#), as events. The calls captured include calls from the SageMaker console and code calls to the SageMaker API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for SageMaker. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to SageMaker, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [Amazon CloudTrail User Guide](#).

By default, log data is stored in CloudWatch Logs indefinitely. However, you can configure how long to store log data in a log group. For information, see [Change Log Data Retention in CloudWatch Logs](#) in the *Amazon CloudWatch Logs User Guide*.

## SageMaker Information in CloudTrail

CloudTrail is enabled on your Amazon account when you create the account. When activity occurs in Amazon SageMaker, that activity is recorded in a CloudTrail event along with other Amazon service events in **Event history**. You can view, search, and download recent events in your Amazon account. For more information, see [Viewing Events with CloudTrail Event History](#).

For an ongoing record of events in your Amazon account, including events for Amazon SageMaker, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all Amazon Regions. The trail logs events from all Regions in the Amazon partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other Amazon services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- [Overview for Creating a Trail](#)
- [CloudTrail Supported Services and Integrations](#)
- [Configuring Amazon SNS Notifications for CloudTrail](#)
- [Receiving CloudTrail Log Files from Multiple Regions](#) and [Receiving CloudTrail Log Files from Multiple Accounts](#)

All SageMaker actions, with the exception of [InvokeEndpoint](#), are logged by CloudTrail and are documented in the [Operations](#). For example, calls to the [CreateTrainingJob](#), [CreateEndpoint](#) and [CreateNotebookInstance](#) actions generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or Amazon Identity and Access Management (IAM) user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another Amazon service.

For more information, see the [CloudTrail userIdentity Element](#).

## Operations Performed by Automatic Model Tuning

SageMaker supports logging non-API service events to your CloudTrail log files for automatic model tuning jobs. These events are related to your tuning jobs but, are not the direct result of a customer request to the public Amazon API. For example, when you create a hyperparameter tuning job by calling [CreateHyperParameterTuningJob](#), SageMaker creates training jobs to evaluate various combinations of hyperparameters to find the best result. Similarly, when you call [StopHyperParameterTuningJob](#) to stop a hyperparameter tuning job, SageMaker might stop any of the associated running training jobs. Non-API events for your tuning jobs are logged to CloudTrail to help you improve governance, compliance, and operational and risk auditing of your Amazon account.

Log entries that result from non-API service events have an `eventType` of `AwsServiceEvent` instead of `AwsApiCall`.

## Understanding SageMaker Log File Entries

A trail is a configuration that enables delivery of events as log files to an S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

The following examples a log entry for the [CreateEndpoint](#) action, which creates an endpoint to deploy a trained model.

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIXDAYQEXAMPLEUMLYNGL",
    "arn": "arn:aws:iam::123456789012:user/intern",
    "accountId": "123456789012",
    "accessKeyId": "ASXIAGXAMPLEQULKNXV",
    "userName": "intern"
  },
  "eventTime": "2018-01-02T13:39:06Z",
  "version": "0.0.1"
}
```

```

    "eventSource": "sagemaker.amazonaws.com",
    "eventName": "CreateEndpoint",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "USER_AGENT",
    "requestParameters": {
        "endpointName": "ExampleEndpoint",
        "endpointConfigName": "ExampleEndpointConfig"
    },
    "responseElements": {
        "endpointArn": "arn:aws:sagemaker:us-west-2:123456789012:endpoint/exampleendpoint"
    },
    "requestID": "6b1b42b9-EXAMPLE",
    "eventID": "a6f85b21-EXAMPLE",
    "eventType": "AwsApiCall",
    "recipientAccountId": "444455556666"
}

```

The following example is a log entry for the `CreateModel` action, which creates one or more containers to host a previously trained model.

```

{
    "eventVersion": "1.05",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIXDAYQEXAMPLEUMLYNGL",
        "arn": "arn:aws:iam::123456789012:user/intern",
        "accountId": "123456789012",
        "accessKeyId": "ASXIAGXEXAMPLEQUEULKNXV",
        "userName": "intern"
    },
    "eventTime": "2018-01-02T15:23:46Z",
    "eventSource": "sagemaker.amazonaws.com",
    "eventName": "CreateModel",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "USER_AGENT",
    "requestParameters": {
        "modelName": "ExampleModel",
        "primaryContainer": {
            "image": "174872318107.dkr.ecr.us-west-2.amazonaws.com/kmeans:latest"
        },
        "executionRoleArn": "arn:aws:iam::123456789012:role/EXAMPLEARN"
    },
    "responseElements": {
        "modelArn": "arn:aws:sagemaker:us-west-2:123456789012:model/barkinghappy2018-01-02t15-23-32-275z-ivrdog"
    },
    "requestID": "417b8dab-EXAMPLE",
    "eventID": "0f2b3e81-EXAMPLE",
    "eventType": "AwsApiCall",
    "recipientAccountId": "444455556666"
}

```

## Automating Amazon SageMaker with Amazon EventBridge

Amazon EventBridge monitors status change events in Amazon SageMaker. EventBridge enables you to automate SageMaker and respond automatically to events such as a training job status change or

endpoint status change. Events from SageMaker are delivered to EventBridge in near real time. You can write simple rules to indicate which events are of interest to you, and what automated actions to take when an event matches a rule. For an example of how to create a rule, see [Schedule a Pipeline with Amazon EventBridge \(p. 2193\)](#).

Some examples of the actions that can be automatically triggered include the following:

- Invoking an Amazon Lambda function
- Invoking Amazon EC2 Run Command
- Relaying the event to Amazon Kinesis Data Streams
- Activating an Amazon Step Functions state machine
- Notifying an Amazon SNS topic or an Amazon SMS queue

#### SageMaker events monitored by EventBridge

- [Training job state change \(p. 2538\)](#)
- [Hyperparameter tuning job state change \(p. 2539\)](#)
- [Transform job state change \(p. 2541\)](#)
- [Endpoint state change \(p. 2541\)](#)
- [Feature group state change \(p. 2542\)](#)
- [Model package state change \(p. 2543\)](#)
- [Pipeline execution state change \(p. 2544\)](#)
- [Pipeline step state change \(p. 2544\)](#)

## Training job state change

Indicates a change in the status of a SageMaker training job.

If the value of `TrainingJobStatus` is `Failed`, the event contains the `FailureReason` field, which provides a description of why the training job failed.

```
{  
    "version": "0",  
    "id": "844e2571-85d4-695f-b930-0153b71dc842",  
    "detail-type": "SageMaker Training Job State Change",  
    "source": "aws.sagemaker",  
    "account": "123456789012",  
    "time": "2018-10-06T12:26:13Z",  
    "region": "us-east-1",  
    "resources": [  
        "arn:aws:sagemaker:us-east-1:123456789012:training-job/kmeans-1"  
    ],  
    "detail": {  
        "TrainingJobName": "89c96cc8-dded-4739-afcc-6f1dc936701d",  
        "TrainingJobArn": "arn:aws:sagemaker:us-east-1:123456789012:training-job/kmeans-1",  
        "TrainingJobStatus": "Completed",  
        "SecondaryStatus": "Completed",  
        "HyperParameters": {  
            "Hyper": "Parameters"  
        },  
        "AlgorithmSpecification": {  
            "TrainingImage": "TrainingImage",  
            "TrainingInputMode": "TrainingInputMode"  
        },  
        "RoleArn": "arn:aws:iam::123456789012:role/SMRole",  
        "FailureReason": "The training job failed because the training algorithm did not converge."  
    }  
}
```

```

    "InputDataConfig": [
        {
            "ChannelName": "Train",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3DataType",
                    "S3Uri": "S3Uri",
                    "S3DataDistributionType": "S3DataDistributionType"
                }
            },
            "ContentType": "ContentType",
            "CompressionType": "CompressionType",
            "RecordWrapperType": "RecordWrapperType"
        }
    ],
    "OutputDataConfig": {
        "KmsKeyId": "KmsKeyId",
        "S3OutputPath": "S3OutputPath"
    },
    "ResourceConfig": {
        "InstanceType": "InstanceType",
        "InstanceCount": 3,
        "VolumeSizeInGB": 20,
        "VolumeKmsKeyId": "VolumeKmsKeyId"
    },
    "VpcConfig": {

    },
    "StoppingCondition": {
        "MaxRuntimeInSeconds": 60
    },
    "CreationTime": "1583831889050",
    "TrainingStartTime": "1583831889050",
    "TrainingEndTime": "1583831889050",
    "LastModifiedTime": "1583831889050",
    "SecondaryStatusTransitions": [
    ],
    "Tags": {
    }
}
}

```

## Hyperparameter tuning job state change

Indicates a change in the status of a SageMaker hyperparameter tuning job.

```
{
    "version": "0",
    "id": "844e2571-85d4-695f-b930-0153b71dcba2",
    "detail-type": "SageMaker HyperParameter Tuning Job State Change",
    "source": "aws.sagemaker",
    "account": "123456789012",
    "time": "2018-10-06T12:26:13Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:sagemaker:us-east-1:123456789012:tuningJob/x"
    ],
    "detail": {
        "HyperParameterTuningJobName": "016bffd3-6d71-4d3a-9710-0a332b2759fc",
        "HyperParameterTuningJobArn": "arn:aws:sagemaker:us-east-1:123456789012:tuningJob/x",
        "TrainingJobDefinition": {

```

```

    "StaticHyperParameters": {},
    "AlgorithmSpecification": {
        "TrainingImage": "trainingImageName",
        "TrainingInputMode": "inputModeFile",
        "MetricDefinitions": [
            {
                "Name": "metricName",
                "Regex": "regex"
            }
        ],
        "RoleArn": "roleArn",
        "InputDataConfig": [
            {
                "ChannelName": "channelName",
                "DataSource": {
                    "S3DataSource": {
                        "S3DataType": "s3DataType",
                        "S3Uri": "s3Uri",
                        "S3DataDistributionType": "s3DistributionType"
                    }
                },
                "ContentType": "contentType",
                "CompressionType": "gz",
                "RecordWrapperType": "RecordWrapper"
            }
        ],
        "VpcConfig": {
            "SecurityGroupIds": [
                "securityGroupIds"
            ],
            "Subnets": [
                "subnets"
            ]
        },
        "OutputDataConfig": {
            "KmsKeyId": "kmsKeyId",
            "S3OutputPath": "s3OutputPath"
        },
        "ResourceConfig": {
            "InstanceType": "instanceType",
            "InstanceCount": 10,
            "VolumeSizeInGB": 500,
            "VolumeKmsKeyId": "volumeKeyId"
        },
        "StoppingCondition": {
            "MaxRuntimeInSeconds": 3600
        }
    },
    "HyperParameterTuningJobStatus": "status",
    "CreationTime": "1583831889050",
    "LastModifiedTime": "1583831889050",
    "TrainingJobStatusCounters": {
        "Completed": 1,
        "InProgress": 0,
        "RetryableError": 0,
        "NonRetryableError": 0,
        "Stopped": 0
    },
    "ObjectiveStatusCounters": {
        "Succeeded": 1,
        "Pending": 0,
        "Failed": 0
    },
    "Tags": {}
}

```

}

## Transform job state change

Indicates a change in the status of a SageMaker batch transform job.

If the value of `TransformJobStatus` is `Failed`, the event contains the `FailureReason` field, which provides a description of why the training job failed.

```
{  
    "version": "0",  
    "id": "844e2571-85d4-695f-b930-0153b71dcb42",  
    "detail-type": "SageMaker Transform Job State Change",  
    "source": "aws.sagemaker",  
    "account": "123456789012",  
    "time": "2018-10-06T12:26:13Z",  
    "region": "us-east-1",  
    "resources": ["arn:aws:sagemaker:us-east-1:123456789012:transform-job/myjob"],  
    "detail": {  
        "TransformJobName": "4b52bd8f-e034-4345-818d-884bdd7c9724",  
        "TransformJobArn": "arn:aws:sagemaker:us-east-1:123456789012:transform-job/myjob",  
        "TransformJobStatus": "another status... GO",  
        "FailureReason": "failed why 1",  
        "ModelName": "i am a beautiful model",  
        "MaxConcurrentTransforms": 5,  
        "MaxPayloadInMB": 10,  
        "BatchStrategy": "Strategizing...",  
        "Environment": {  
            "environment1": "environment2"  
        },  
        "TransformInput": {  
            "DataSource": {  
                "S3DataSource": {  
                    "S3DataType": "s3DataType",  
                    "S3Uri": "s3Uri"  
                }  
            },  
            "ContentType": "content type",  
            "CompressionType": "compression type",  
            "SplitType": "split type"  
        },  
        "TransformOutput": {  
            "S3OutputPath": "s3Uri",  
            "Accept": "accept",  
            "AssembleWith": "assemblyType",  
            "KmsKeyId": "kmsKeyId"  
        },  
        "TransformResources": {  
            "InstanceType": "instanceType",  
            "InstanceCount": 3  
        },  
        "CreationTime": "2018-10-06T12:26:13Z",  
        "TransformStartTime": "2018-10-06T12:26:13Z",  
        "TransformEndTime": "2018-10-06T12:26:13Z",  
        "Tags": {}  
    }  
}
```

## Endpoint state change

Indicates a change in the status of a SageMaker hosted real-time inference endpoint.

The following shows an event with an endpoint in the IN\_SERVICE state.

```
{
    "version": "0",
    "id": "d2921b5a-b0ad-cace-a8e3-0f159d018e06",
    "detail-type": "SageMaker Endpoint State Change",
    "source": "aws.sagemaker",
    "account": "123456789012",
    "time": "1583831889050",
    "region": "us-west-2",
    "resources": [
        "arn:aws:sagemaker:us-west-2:123456789012:endpoint/myendpoint"
    ],
    "detail": {
        "EndpointName": "MyEndpoint",
        "EndpointArn": "arn:aws:sagemaker:us-west-2:123456789012:endpoint/myendpoint",
        "EndpointConfigName": "MyEndpointConfig",
        "ProductionVariants": [
            {
                "DesiredWeight": 1.0,
                "DesiredInstanceCount": 1.0
            }
        ],
        "EndpointStatus": "IN_SERVICE",
        "CreationTime": 1592411992203.0,
        "LastModifiedTime": 1592411994287.0,
        "Tags": {
        }
    }
}
```

## Feature group state change

Indicates a change either in the `FeatureGroupStatus` or the `OfflineStoreStatus` of a SageMaker feature group.

```
{
    "version": "0",
    "id": "93201303-abdb-36a4-1b9b-4c1c3e3671c0",
    "detail-type": "SageMaker Feature Group State Change",
    "source": "aws.sagemaker",
    "account": "123456789012",
    "time": "2021-01-26T01:22:01Z",
    "region": "us-east-1",
    "resources": [
        "arn:aws:sagemaker:us-east-1:123456789012:feature-group/sample-feature-group"
    ],
    "detail": {
        "FeatureGroupArn": "arn:aws:sagemaker:us-east-1:123456789012:feature-group/sample-feature-group",
        "FeatureGroupName": "sample-feature-group",
        "RecordIdentifierFeatureName": "RecordIdentifier",
        "EventTimeFeatureName": "EventTime",
        "FeatureDefinitions": [
            {
                "FeatureName": "RecordIdentifier",
                "FeatureType": "Integral"
            },
            {
                "FeatureName": "EventTime",
                "FeatureType": "Fractional"
            }
        ]
    }
}
```

```

        ],
        "CreationTime": 1611624059000,
        "OnlineStoreConfig": {
            "EnableOnlineStore": true
        },
        "OfflineStoreConfig": {
            "S3StorageConfig": {
                "S3Uri": "s3://offline/s3/uri"
            },
            "DisableGlueTableCreation": false,
            "DataCatalogConfig": {
                "TableName": "sample-feature-group-1611624059",
                "Catalog": "AwsDataCatalog",
                "Database": "sagemaker_featurestore"
            }
        },
        "RoleArn": "arn:aws:iam::123456789012:role/SageMakerRole",
        "FeatureGroupStatus": "Active",
        "Tags": {}
    }
}

```

## Model package state change

Indicates a change in the status of a SageMaker model package.

```
{
    "version": "0",
    "id": "844e2571-85d4-695f-b930-0153b71dcb42",
    "detail-type": "SageMaker Model Package State Change",
    "source": "aws.sagemaker",
    "account": "123456789012",
    "time": "2021-02-24T17:00:14Z",
    "region": "us-east-2",
    "resources": [
        "arn:aws:sagemaker:us-east-2:123456789012:model-package/versionedmp-p-idy6c3e1fiqj/2"
    ],
    "source": [
        "aws.sagemaker"
    ],
    "detail": {
        "ModelPackageGroupName": "versionedmp-p-idy6c3e1fiqj",
        "ModelPackageVersion": 2,
        "ModelPackageArn": "arn:aws:sagemaker:us-east-2:123456789012:model-package/versionedmp-p-idy6c3e1fiqj/2",
        "CreationTime": "2021-02-24T17:00:14Z",
        "InferenceSpecification": {
            "Containers": [
                {
                    "Image": "257758044811.dkr.ecr.us-east-2.amazonaws.com/sagemaker-xgboost:1.0-1-cpu-py3",
                    "ImageDigest": "sha256:4dc8a7e4a010a19bb9e0a6b063f355393f6e623603361bd8b105f554d4f0c004",
                    "ModelDataURL": "s3://sagemaker-project-p-idy6c3e1fiqj/versionedmp-p-idy6c3e1fiqj/AbaloneTrain/pipelines-4r83jejmhorv-TrainAbaloneModel-xw869y8C4a/output/model.tar.gz"
                }
            ],
            "SupportedContentTypes": [
                "text/csv"
            ],
            "SupportedResponseMIMETypes": [
                "text/csv"
            ]
        }
    }
}
```

```
{
    "ModelPackageStatus": "Completed",
    "ModelPackageStatusDetails": {
        "ValidationStatuses": [],
        "ImageScanStatuses": []
    },
    "CertifyForMarketplace": false,
    "ModelApprovalStatus": "Rejected",
    "MetadataProperties": {
        "GeneratedBy": "arn:aws:sagemaker:us-east-2:123456789012:pipeline/versionedmp-p-  
idy6c3e1fiqj/execution/4r83jejmhorv"
    },
    "ModelMetrics": {
        "ModelQuality": {
            "Statistics": {
                "ContentType": "application/json",
                "S3Uri": "s3://sagemaker-project-p-  
idy6c3e1fiqj/versionedmp-p-  
idy6c3e1fiqj/script-2021-02-24-10-55-15-413/output/evaluation/evaluation.json"
            }
        }
    },
    "LastModifiedTime": "2021-02-24T17:00:14Z"
}
}
```

## Pipeline execution state change

Indicates a change in the status of a SageMaker pipeline execution.

```
{
    "version": "0",
    "id": "315c1398-40ff-a850-213b-158f73kd93ir",
    "detail-type": "SageMaker Model Building Pipeline Execution Status Change",
    "source": "aws.sagemaker",
    "account": "123456789012",
    "time": "2021-03-15T16:10:11Z",
    "region": "us-east-1",
    "resources": ["arn:aws:sagemaker:us-east-1:123456789012:pipeline/myPipeline-123",
    "arn:aws:sagemaker:us-east-1:123456789012:pipeline/myPipeline-123/execution/  
p4jn9xou8a8s"],
    "detail": {
        "pipelineExecutionDisplayName": "SomeDisplayName",
        "currentPipelineExecutionStatus": "Succeeded",
        "previousPipelineExecutionStatus": "Executing",
        "executionStartTime": "2021-03-15T16:03:13Z",
        "executionEndTime": "2021-03-15T16:10:10Z",
        "pipelineExecutionDescription": "SomeDescription",
        "pipelineArn": "arn:aws:sagemaker:us-east-1:123456789012:pipeline/myPipeline-123",
        "pipelineExecutionArn": "arn:aws:sagemaker:us-east-1:123456789012:pipeline/  
myPipeline-123/execution/p4jn9xou8a8s"
    }
}
```

## Pipeline step state change

Indicates a change in the status of a SageMaker pipeline step.

If there is a cache hit, the event contains the `cacheHitResult` field.

If the value of `currentStepStatus` is `Failed`, the event contains the `failureReason` field, which provides a description of why the step failed.

```
{  
    "version": "0",  
    "id": "ea37ccbb-5e2b-05e9-4073-1daazc940304",  
    "detail-type": "SageMaker Model Building Pipeline Execution Step Status Change",  
    "source": "aws.sagemaker",  
    "account": "123456789012",  
    "time": "2021-03-15T16:10:10Z",  
    "region": "us-east-1",  
    "resources": ["arn:aws:sagemaker:us-east-1:123456789012:pipeline/myPipeline-123",  
                 "arn:aws:sagemaker:us-east-1:123456789012:pipeline/myPipeline-123/execution/  
                  p4jn9xou8a8s"],  
    "detail": {  
        "metadata": {  
            "processingJob": {  
                "arn": "arn:aws:sagemaker:us-east-1:123456789012:processing-job/pipelines-  
                  p4jn9xou8a8s-myprocessingstep1-tmgxry49ug"  
            }  
        },  
        "stepStartTime": "2021-03-15T16:03:14Z",  
        "stepEndTime": "2021-03-15T16:10:09Z",  
        "stepName": "myprocessingstep1",  
        "stepType": "Processing",  
        "previousStepStatus": "Executing",  
        "currentStepStatus": "Succeeded",  
        "pipelineArn": "arn:aws:sagemaker:us-east-1:123456789012:pipeline/myPipeline-123",  
        "pipelineExecutionArn": "arn:aws:sagemaker:us-east-1:123456789012:pipeline/  
          myPipeline-123/execution/p4jn9xou8a8s"  
    }  
}
```

For more information about the status values and their meanings for SageMaker jobs, endpoints, and pipelines, see the following links:

- [AlgorithmStatus](#)
- [EndpointStatus](#)
- [FeatureGroupStatus](#)
- [HyperParameterTuningJobStatus](#)
- [LabelingJobStatus](#)
- [ModelPackageStatus](#)
- [NotebookInstanceStatus](#)
- [PipelineExecutionStatus](#)
- [StepStatus](#)
- [ProcessingJobStatus](#)
- [TrainingJobStatus](#)
- [TransformJobStatus](#)

For more information, see the [Amazon EventBridge User Guide](#).

# Availability Zones

For the Amazon Regions supported by Amazon SageMaker and the Amazon Elastic Compute Cloud (Amazon EC2) instance types that are available in each Region, see [Amazon SageMaker Pricing](#).

Each Amazon Region is divided into sub-regions known as Availability Zones. For a given Region, the Availability Zones in that Region don't always contain the same instance types supported by SageMaker.

Amazon SageMaker Studio is available in all the Amazon Regions supported by Amazon SageMaker except the Amazon GovCloud (US) Regions. In the supported Regions, Studio is available in the same Availability Zones as notebook instances.

For more information, see [Regions and Availability Zones](#) in the *Amazon EC2 User Guide* and [AZ IDs for Your Resources](#) in the *Amazon RAM User Guide*.

The following list provides links to the Availability Zone tables for each Amazon Region supported by SageMaker. Each linked table has a column for each Availability Zone in the Region. For each Availability Zone, the SageMaker components that support each instance type are shown.

## Availability Zone Tables

- [US East \(Ohio\) us-east-2](#)
- [US East \(N. Virginia\) us-east-1](#)
- [US West \(N. California\) us-west-1](#)
- [US West \(Oregon\) us-west-2](#)
- [Asia Pacific \(Hong Kong\) ap-east-1](#)
- [Asia Pacific \(Mumbai\) ap-south-1](#)
- [Asia Pacific \(Seoul\) ap-northeast-2](#)
- [Asia Pacific \(Singapore\) ap-southeast-1](#)
- [Asia Pacific \(Sydney\) ap-southeast-2](#)
- [Asia Pacific \(Tokyo\) ap-northeast-1](#)
- [Canada \(Central\) ca-central-1](#)
- [EU \(Frankfurt\) eu-central-1](#)
- [EU \(Ireland\) eu-west-1](#)
- [EU \(London\) eu-west-2](#)
- [EU \(Paris\) eu-west-3](#)
- [EU \(Stockholm\) eu-north-1](#)
- [Middle East \(Bahrain\) me-south-1](#)
- [South America \(Sao Paulo\) sa-east-1](#)
- [Amazon GovCloud \(US-Gov-West\) us-gov-west-1](#)
- [US ISO East us-iso-east-1](#)

The following Amazon Regions are supported by Amazon SageMaker. Availability Zones tables are not available at this time. For more information, contact Amazon Support.

- [Africa \(Cape Town\) af-south-1](#)
- [EU \(Milan\) eu-south-1](#)

# API Reference Guide for Amazon SageMaker

## Overview

Amazon SageMaker provides APIs, SDKs, and a command line interface that you can use to create and manage notebook instances and train and deploy models.

- [Amazon SageMaker Python SDK](#) (Recommended)
- [Amazon SageMaker API Reference](#)
- [Amazon Augmented AI API Reference](#)
- [Amazon Command Line Interface](#)
- [Amazon SDK for .NET](#)
- [Amazon SDK for C++](#)
- [Amazon SDK for Go](#)
- [Amazon SDK for Java](#)
- [Amazon SDK for JavaScript](#)
- [Amazon SDK for PHP](#)
- [Amazon SDK for Python \(Boto\)](#)
- [Amazon SDK for Ruby](#)
- [Amazon SageMaker Spark](#)

You can also get code examples from the Amazon SageMaker example notebooks GitHub repository.

- [Example notebooks](#)

## Programming Model for Amazon SageMaker

Making API calls directly from code is cumbersome, and requires you to write code to authenticate your requests. Amazon SageMaker provides the following alternatives:

- **Use the SageMaker console**—With the console, you don't write any code. You use the console UI to start model training or deploy a model. The console works well for simple jobs, where you use a built-in training algorithm and you don't need to preprocess training data.
- **Modify the example Jupyter notebooks**—SageMaker provides several Jupyter notebooks that train and deploy models using specific algorithms and datasets. Start with a notebook that has a suitable algorithm and modify it to accommodate your data source and specific needs.
- **Write model training and inference code from scratch**—SageMaker provides multiple Amazon SDK languages (listed in the overview) and the [Amazon SageMaker Python SDK](#), a high-level Python library that you can use in your code to start model training jobs and deploy the resulting models.

- **The SageMaker Python SDK**—This Python library simplifies model training and deployment. In addition to authenticating your requests, the library abstracts platform specifics by providing simple methods and default parameters. For example:
  - To deploy your model, you call only the `deploy()` method. The method creates a SageMaker model artifact, an endpoint configuration, then deploys the model on an endpoint.
  - If you use a custom framework script for model training, you call the `fit()` method. The method creates a .gzip file of your script, uploads it to an Amazon S3 location, and then runs it for model training, and other tasks. For more information, see [Use Machine Learning Frameworks, Python, and R with Amazon SageMaker \(p. 16\)](#).
- **The Amazon SDKs** – The SDKs provide methods that correspond to the SageMaker API (see [Operations](#)). Use the SDKs to programmatically start a model training job and host the model in SageMaker. SDK clients authenticate your requests by using your access keys, so you don't need to write authentication code. They are available in multiple languages and platforms. For more information, see the preceeding list in the overview.

In [Get Started with Amazon SageMaker \(p. 34\)](#), you train and deploy a model using an algorithm provided by SageMaker. That exercise shows how to use both of these libraries. For more information, see [Get Started with Amazon SageMaker \(p. 34\)](#).

- **Integrate SageMaker into your Apache Spark workflow**—SageMaker provides a library for calling its APIs from Apache Spark. With it, you can use SageMaker-based estimators in an Apache Spark pipeline. For more information, see [Use Apache Spark with Amazon SageMaker \(p. 17\)](#).

# Document History for Amazon SageMaker

update-history-change	update-history-description	update-history-date
<a href="#">SageMaker Clarify update 1.0.5</a>	Instance explanations in the CSV file for models with multiple outputs now map correctly to the right columns in v1.0.5. Also, improved numerical handling of models returning 0 and 1 as class probabilities when using logits in Kernel SHAP.	April 22, 2021
<a href="#">New features re:Invent 2020 (p. 2549)</a>	<a href="#">Amazon SageMaker Model Building Pipelines, Automate MLOps with SageMaker Projects, SageMaker Edge Manager, SageMaker Clarify, SageMaker Data Wrangler, SageMaker Feature Store, SageMaker Studio JumpStart, Register and Deploy Models with Model Registry, SageMaker Distributed, Deep Profiling with SageMaker Debugger</a>	December 1, 2020
<a href="#">Studio Notebooks (p. 2549)</a>	<a href="#">SageMaker Studio Notebooks</a>	April 28, 2020
<a href="#">New features re:Invent 2019 (p. 2549)</a>	<a href="#">SageMaker Studio, SageMaker Studio Notebooks (preview), SageMaker Experiments, SageMaker Autopilot, SageMaker Debugger, SageMaker Model Monitor</a>	December 3, 2019
<a href="#">New features re:Invent 2018 (p. 2549)</a>	<a href="#">Amazon SageMaker Ground Truth, Using Elastic Inference in Amazon SageMaker, SageMaker Resources in Amazon Web Services Marketplace, SageMaker Inference Pipelines, SageMaker Neo, Manage Machine Learning Experiments with Search, Use Reinforcement Learning in Amazon SageMaker, Associating Git Repositories with Amazon SageMaker Notebook Instances, Semantic Segmentation, Using Augmented Manifest Files in TrainingJobs</a>	November 28, 2018

<a href="#">Configuring notebook instances (p. 2549)</a>	You can use shell scripts to configure notebook instances when you create or start them. For more information, see <a href="#">Customize a Notebook Instance</a> .	May 1, 2018
<a href="#">Application Auto Scaling support (p. 2549)</a>	Amazon SageMaker now supports Application Auto Scaling for production variants. For information, see <a href="#">Automatically Scaling SageMaker Models</a>	February 28, 2018
<a href="#">TensorFlow 1.5 and MXNet 1.0 support (p. 2549)</a>	Amazon SageMaker Deep Learning containers now support TensorFlow 1.5 and Apache MXNet 1.0.	February 27, 2018
<a href="#">BlazingText algorithm (p. 2549)</a>	Amazon SageMaker now supports the <a href="#">BlazingText</a> algorithm.	January 18, 2018
<a href="#">KMS encryption (p. 2549)</a>	Amazon SageMaker now supports KMS encryption for hosting instances and training model artifacts at rest.	January 17, 2018
<a href="#">CloudTrail support (p. 2549)</a>	Amazon SageMaker now supports <a href="#">logging with Amazon CloudTrail</a> .	January 11, 2018
<a href="#">DeepAR Forecasting algorithm (p. 2549)</a>	Amazon SageMaker now supports the <a href="#">DeepAR</a> algorithm for time series forecasting.	January 8, 2018

# Amazon glossary

For the latest Amazon terminology, see the [Amazon glossary](#) in the *Amazon General Reference*.