
Rekognition

Custom Labels Guide



Rekognition: Custom Labels Guide

Copyright © 2022 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What Is Amazon Rekognition Custom Labels?	1
Key Benefits	1
Are You a First-Time Amazon Rekognition Custom Labels User?	2
Choosing to Use Amazon Rekognition Custom Labels	2
Amazon Rekognition Label Detection	2
Amazon Rekognition Custom Labels	2
Setting up Amazon Rekognition Custom Labels	4
Step 1: Create an AWS account	4
Step 2: Create an IAM administrator user and group	4
Step 3: Set Up the AWS CLI and AWS SDKs	5
Step 4: Set Up Permissions	6
Allowing console access	6
Accessing external Amazon S3 Buckets	7
Policy updates for using the AWS SDK	8
Step 5: Create the console bucket	8
Step 6: (Optional) Encrypt training files	9
Decrypting files encrypted with AWS Key Management Service	9
Encrypting copied training and test images	9
Step 7: (Optional) Associate prior datasets	10
Using a prior dataset as a test dataset	10
Getting started	11
Tutorial videos	11
Example projects	11
Image classification	12
Multi-label image classification	12
Brand detection	12
Object localization	13
Using the example projects	13
Creating the example project	13
Training the model	14
Using the model	14
Next steps	14
Step 1: Choose an example project	14
Step 2: Train your model	16
Step 3: Start your model	18
Step 4: Analyze an image with your model	19
Getting an example image	22
Step 5: Stop your model	24
Step 6: Next steps	25
Understanding Amazon Rekognition Custom Labels	27
Decide your model type	27
Find objects, scenes, and concepts	27
Find object locations	28
Find the location of brands	29
Create a model	29
Create a project	29
Create training and test datasets	30
Train your model	31
Improve your model	31
Evaluate your model	31
Improve your model	32
Start your model	32
Start your model (console)	32
Start your model	32

Analyze an image	32
Stop your model	33
Stop your model (SDK)	33
Stop your model (SDK)	33
Tutorial: Classifying images	34
Step 1: Collect your images	34
Step 2: Decide your classes	35
Step 3: Create a project	35
Step 4: Create training and test datasets	36
Step 5: Add labels to the project	39
Step 6: Assign image-level labels to training and test datasets	40
Step 7: Train your model	41
Step 8: Start your model	44
Step 9: Analyze an image with your model	45
Step 10: Stop your model	48
Creating a model	50
Creating a project	50
Creating a Project (Console)	50
Creating a project (SDK)	51
Creating datasets	54
Purposing datasets	55
Preparing images	58
Creating datasets (Console)	59
Creating datasets (SDK)	65
Debugging datasets	95
Labeling images	99
Training a model	106
Training a model (Console)	107
Training a model (SDK)	109
Debugging model training	116
Terminal errors	116
Non terminal JSON line validation errors	118
Understanding the manifest summary	118
Understanding training and testing validation result manifests	121
Getting the validation results	124
Fixing training errors	126
Terminal manifest file errors	127
Terminal manifest content errors	129
Non-Terminal JSON Line Validation Errors	135
Improving a trained model	152
Metrics for evaluating your model	152
Evaluating model performance	152
Assumed threshold	153
Precision	153
Recall	154
F1	154
Using metrics	154
Accessing evaluation metrics (Console)	155
Accessing evaluation metrics (SDK)	156
Summary file	157
Evaluation manifest snapshot	158
Accessing the summary file and evaluation manifest snapshot (SDK)	161
Reference: Summary File	161
Improving a model	163
Data	163
Reducing false positives (better precision)	163
Reducing false negatives (better recall)	164

Running a trained model	165
Inference units	165
Managing throughput with inference units	166
Availability Zones	167
Starting a model	167
Starting or stopping a model (Console)	168
Starting a model (SDK)	168
Stopping a model	175
Stopping a model (Console)	175
Stopping a model (SDK)	176
Analyzing an image	182
DetectCustomLabels operation request	199
DetectCustomLabels operation response	199
Managing resources	200
Managing a project	200
Deleting a project	200
Describing a project (SDK)	207
Creating a project with AWS CloudFormation	211
Managing datasets	211
Adding a dataset	212
Adding more images	218
Describing a dataset (SDK)	224
Listing dataset entries (SDK)	227
Distributing a training dataset (SDK)	231
Deleting a dataset	238
Creating a manifest file	242
Managing a model	268
Deleting a model	268
Tagging a model	274
Describing a model (SDK)	279
Copying a model (SDK)	284
Examples	309
Model feedback solution	309
Amazon Rekognition Custom Labels demonstration	309
Video analysis	310
Creating an AWS Lambda function	311
Creating a manifest file from a CSV file	313
Security	319
Securing Amazon Rekognition Custom Labels projects	319
Securing DetectCustomLabels	320
AWS managed policy: AmazonRekognitionCustomLabelsFullAccess	320
AmazonRekognitionCustomLabelsFullAccess	321
Policy updates	322
Guidelines and quotas	323
Supported Regions	323
Quotas	323
Training	323
Testing	323
Detection	324
Model copying	324
API reference	325
Training your model	331
Projects	331
Project Policies	331
Datasets	332
Models	332
Tags	332

Using your model	332
Document history	333
AWS glossary	336

What Is Amazon Rekognition Custom Labels?

With Amazon Rekognition Custom Labels, you can identify the objects and scenes in images that are specific to your business needs. For example, you can find your logo in social media posts, identify your products on store shelves, classify machine parts in an assembly line, distinguish healthy and infected plants, or detect animated characters in videos.

Developing a custom model to analyze images is a significant undertaking that requires time, expertise, and resources. It often takes months to complete. Additionally, it can require thousands or tens of thousands of hand-labeled images to provide the model with enough data to accurately make decisions. Generating this data can take months to gather, and can require large teams of labelers to prepare it for use in machine learning.

Amazon Rekognition Custom Labels extends Amazon Rekognition's existing capabilities, which are already trained on tens of millions of images across many categories. Instead of thousands of images, you can upload a small set of training images (typically a few hundred images or less) that are specific to your use case. You can do this by using the easy-to-use console. If your images are already labeled, Amazon Rekognition Custom Labels can begin training a model in a short time. If not, you can label the images directly within the labeling interface, or you can use Amazon SageMaker Ground Truth to label them for you.

After Amazon Rekognition Custom Labels begins training from your image set, it can produce a custom image analysis model for you in just a few hours. Behind the scenes, Amazon Rekognition Custom Labels automatically loads and inspects the training data, selects the right machine learning algorithms, trains a model, and provides model performance metrics. You can then use your custom model through the Amazon Rekognition Custom Labels API and integrate it into your applications.

Key Benefits

Simplified data labeling

The Amazon Rekognition Custom Labels console provides a visual interface to make labeling your images fast and simple. The interface allows you to apply a label to the entire image. You can also identify and label specific objects in images using bounding boxes with a click-and-drag interface. Alternately, if you have a large dataset, you can use [Amazon SageMaker Ground Truth](#) to efficiently label your images at scale.

Automated machine learning

No machine learning expertise is required to build your custom model. Amazon Rekognition Custom Labels includes automated machine learning (AutoML) capabilities that take care of the machine learning for you. When the training images are provided, Amazon Rekognition Custom Labels can automatically load and inspect the data, select the right machine learning algorithms, train a model, and provide model performance metrics.

Simplified model evaluation, inference, and feedback

You evaluate your custom model's performance on your test set. For every image in the test set, you can see the side-by-side comparison of the model's prediction vs. the label assigned. You can also review detailed performance metrics such as precision, recall, F1 scores, and confidence scores. You can start

using your model immediately for image analysis, or you can iterate and retrain new versions with more images to improve performance. After you start using your model, you track your predictions, correct any mistakes, and use the feedback data to retrain new model versions and improve performance.

Are You a First-Time Amazon Rekognition Custom Labels User?

If you're a first-time user of Amazon Rekognition Custom Labels, we recommend that you read the following sections in order:

1. [Setting up Amazon Rekognition Custom Labels \(p. 4\)](#) – In this section, you set your account details.
2. [Understanding Amazon Rekognition Custom Labels \(p. 27\)](#) – In this section, you learn about the workflow for creating a model.
3. [Getting started with Amazon Rekognition Custom Labels \(p. 11\)](#) – In this section, you train a model using example projects created by Amazon Rekognition Custom Labels.
4. [Tutorial: Classifying images \(p. 34\)](#) – In this section, you learn how to train a model that classifies images with datasets that you create.

Choosing to Use Amazon Rekognition Custom Labels

You can use Amazon Rekognition Custom Labels to find objects, scenes, and concepts in images by using a machine learning model that you create.

Note

Amazon Rekognition Custom Labels is not designed for analyzing faces, detecting text, or finding unsafe image content in images. To perform these tasks, you can use Amazon Rekognition Image. For more information, see [What Is Amazon Rekognition](#).

Amazon Rekognition Label Detection

Customers across media and entertainment, advertising and marketing, industrial, agricultural, and other segments need to identify, classify, and search for important objects, scenes, and other concepts in their images and videos—at scale. Amazon Rekognition's label detection feature makes it fast and easy to detect thousands of common objects (such as cars and trucks, corn and tomatoes, and basketballs and soccer balls) within images and video by using computer vision and machine learning technologies.

However, you can't find specialized objects using Amazon Rekognition label detection. For example, sports leagues want to identify team logos on player jerseys and helmets during game footage, media networks would like to detect specific sponsor logos to report on advertiser coverage, manufacturers need to distinguish between specific machine parts or products in an assembly line to monitor quality, and other customers want to identify cartoon characters in a media library, or locate products of a specific brand on retail shelves. Amazon Rekognition labels don't help you detect these specialized objects.

Amazon Rekognition Custom Labels

You can use Amazon Rekognition Custom Labels to find objects and scenes that are unique to your business needs. Amazon Rekognition Custom Labels supports use cases such as logos, objects, and

scenes. You can use it to perform image classification (image level predictions) or detection (object/bounding box level predictions).

For example, while you can find plants and leaves by using Amazon Rekognition label detection, you need Amazon Rekognition Custom Labels to distinguish between healthy, damaged, and infected plants. Similarly, Amazon Rekognition label detection can identify images with machine parts. However, to identify specific machine parts such as a *turbocharger* or a *torque converter*, you need to use Amazon Rekognition Custom Labels. The following are examples of how you can use Amazon Rekognition Custom Labels.

- Detect logos
- Find animation characters
- Find your products
- Identify machine parts
- Classify agricultural produce quality (such as rotten, ripe, or raw)

Setting up Amazon Rekognition Custom Labels

In this section, you sign up for an AWS account and then create an IAM user, a security group, and optionally set up access to external Amazon S3 buckets used by the Amazon Rekognition Custom Labels console and SDK.

For information about the AWS Regions that support Amazon Rekognition Custom Labels, see [Amazon Rekognition Endpoints and Quotas](#).

Topics

- [Step 1: Create an AWS account \(p. 4\)](#)
- [Step 2: Create an IAM administrator user and group \(p. 4\)](#)
- [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#)
- [Step 4: Set up Amazon Rekognition Custom Labels permissions \(p. 6\)](#)
- [Step 5: Create the console bucket \(p. 8\)](#)
- [Step 6: \(Optional\) Encrypt training files \(p. 9\)](#)
- [\(Optional\) Step 7: Associate prior datasets with new projects \(p. 10\)](#)

Step 1: Create an AWS account

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

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

To create an AWS account

1. Open <https://portal.aws.amazon.com/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.

When you sign up for an AWS account, an *AWS account root user* is created. The root user has access to all AWS services and resources in the account. As a security best practice, [assign administrative access to an administrative user](#), and use only the root user to perform [tasks that require root user access](#).

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

Step 2: Create an IAM administrator user and group

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

to the AWS Management Console by using the email address and password that you used to create the account gives you complete access to all of the AWS resources in your account.

We strongly recommend that you do *not* use the root user for everyday tasks, even the administrative ones. Instead, adhere to the best practice in [Create Individual IAM Users](#), which is to create an AWS 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 and sign in to the console

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

Note

An IAM user with administrator permissions has unrestricted access to the AWS services in your account. You can restrict permissions as necessary. The code examples in this guide assume that you have a user with the AmazonRekognitionFullAccess permissions. You also have to provide permissions to access the console. For more information, see [Step 4: Set up Amazon Rekognition Custom Labels permissions \(p. 6\)](#).

2. Sign in to the AWS Management Console.

To sign in to the AWS Management Console as a IAM user, you must use a special URL. For more information, see [How Users Sign In to Your Account](#) in the *IAM User Guide*.

Step 3: Set Up the AWS CLI and AWS SDKs

The following steps show you how to install the AWS Command Line Interface (AWS CLI) and AWS SDKs that the examples in this documentation use. There are a number of different ways to authenticate AWS SDK calls. The examples in this guide assume that you're using a default credentials profile for calling AWS CLI commands and AWS SDK API operations.

For a list of available AWS Regions, see [Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

Follow the steps to download and configure the AWS SDKs.

To set up the AWS CLI and the AWS SDKs

1. Download and install the [AWS CLI](#) and the AWS SDKs that you want to use. This guide provides examples for the AWS CLI, Java, and Python. For information about installing AWS SDKs, see [Tools for Amazon Web Services](#).
2. Create an access key for the user you created in [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - a. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
 - b. In the navigation pane, choose **Users**.
 - c. Choose the name of the user you created in [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - d. Choose the **Security credentials** tab.
 - e. Choose **Create access key**. Then choose **Download .csv file** to save the access key ID and secret access key to a CSV file on your computer. Store the file in a secure location. You will not have access to the secret access key again after this dialog box closes. After you have downloaded the CSV file, choose **Close**.
3. If you have installed the AWS CLI, you can configure the credentials and Region for most AWS SDKs by entering `aws configure` at the command prompt. Otherwise, use the following instructions.

4. On your computer, navigate to your home directory, and create an `.aws` directory. On Unix-based systems, such as Linux or macOS, this is in the following location:

```
~/.aws
```

On Windows, this is in the following location:

```
%HOMEPATH%\aws
```

5. In the `.aws` directory, create a new file named `credentials`.
6. Open the `credentials` CSV file that you created in step 2. Copy its contents into the `credentials` file using the following format:

```
[default]
aws_access_key_id = your_access_key_id
aws_secret_access_key = your_secret_access_key
```

Substitute your access key ID and secret access key for `your_access_key_id` and `your_secret_access_key`.

7. Save the `Credentials` file and delete the CSV file.
8. In the `.aws` directory, create a new file named `config`.
9. Open the `config` file and enter your Region in the following format.

```
[default]
region = your_aws_region
```

Substitute your desired AWS Region (for example, `us-west-2`) for `your_aws_region`.

Note

If you don't select a Region, then `us-east-1` is used by default.

10. Save the `config` file.

Step 4: Set up Amazon Rekognition Custom Labels permissions

To use the Amazon Rekognition you need add to have appropriate permissions. If you want to store your training files in a bucket other than the console bucket, you need additional permissions.

Topics

- [Allowing console access \(p. 6\)](#)
- [Accessing external Amazon S3 Buckets \(p. 7\)](#)
- [Policy updates for using the AWS SDK \(p. 8\)](#)

Allowing console access

The Identity and Access Management (IAM) user or group that uses the Amazon Rekognition Custom Labels consoles needs the following IAM policy that covers Amazon S3, SageMaker Ground Truth, and Amazon Rekognition Custom Labels. To allow console access, use the following IAM policy. For information about adding IAM policies, see [Creating IAM Policies](#).

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "s3>ListBucket",  
        "s3>ListAllMyBuckets"  
      ],  
      "Resource": "*"  
    },  
    {  
      "Sid": "s3Policies",  
      "Effect": "Allow",  
      "Action": [  
        "s3>ListBucket",  
        "s3>CreateBucket",  
        "s3>GetBucketAcl",  
        "s3>GetBucketLocation",  
        "s3>GetObject",  
        "s3>GetObjectAcl",  
        "s3>GetObjectVersion",  
        "s3>GetObjectTagging",  
        "s3>GetBucketVersioning",  
        "s3>GetObjectVersionTagging",  
        "s3>PutBucketCORS",  
        "s3>PutLifecycleConfiguration",  
        "s3>PutBucketPolicy",  
        "s3>PutObject",  
        "s3>PutObjectTagging",  
        "s3>PutBucketVersioning",  
        "s3>PutObjectVersionTagging"  
      ],  
      "Resource": [  
        "arn:aws:s3:::custom-labels-console-*",  
        "arn:aws:s3:::rekognition-video-console-*"  
      ]  
    },  
    {  
      "Sid": "rekognitionPolicies",  
      "Effect": "Allow",  
      "Action": [  
        "rekognition:/*"  
      ],  
      "Resource": "*"  
    },  
    {  
      "Sid": "groundTruthPolicies",  
      "Effect": "Allow",  
      "Action": [  
        "groundtruthlabeling:/*"  
      ],  
      "Resource": "*"  
    }  
  ]  
}
```

Accessing external Amazon S3 Buckets

When you first open the Amazon Rekognition Custom Labels console in a new AWS Region, Amazon Rekognition Custom Labels creates a bucket (console bucket) that's used to store project files. Alternatively, you can use your own Amazon S3 bucket (external bucket) to upload the images or

manifest file to the console. To use an external bucket, add the following policy block to the preceding policy. Replace `my-bucket` with the name of the bucket.

```
{  
  "Sid": "s3ExternalBucketPolicies",  
  "Effect": "Allow",  
  "Action": [  
    "s3:GetBucketAcl",  
    "s3:GetBucketLocation",  
    "s3:GetObject",  
    "s3:GetObjectAcl",  
    "s3:GetObjectVersion",  
    "s3:GetObjectTagging",  
    "s3>ListBucket",  
    "s3:PutObject"  
  ],  
  "Resource": [  
    "arn:aws:s3:::my-bucket*"  
  ]  
}
```

Policy updates for using the AWS SDK

To use the AWS SDK with the latest release of Amazon Rekognition Custom Labels, you no longer need to give Amazon Rekognition Custom Labels permissions to access the Amazon S3 bucket that contains your training and testing images. If you have previously added permissions, You don't need to remove them. If you choose to, remove any policy from the bucket where the service for the principal is `rekognition.amazonaws.com`. For example:

```
"Principal": {  
  "Service": "rekognition.amazonaws.com"  
}
```

For more information, see [Using bucket policies](#)

Step 5: Create the console bucket

You use an Amazon Rekognition Custom Labels project to create and manage your models. When you first open the Amazon Rekognition Custom Labels console in a new AWS Region, Amazon Rekognition Custom Labels creates an Amazon S3 bucket (console bucket) to store your projects. You should note the console bucket name somewhere where you can refer to it later because you might need to use the bucket name in AWS SDK operations or console tasks, such as creating a dataset.

The format of the bucket name is `custom-labels-console-<region>-<random value>`. The random value ensures that there isn't a collision between bucket names.

To create the console bucket

1. Ensure that the IAM user or group you are using has the correct permissions. For more information, see [Allowing console access \(p. 6\)](#).
2. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
3. Choose **Get started**.
4. If this is the first time that you've opened the console in the current AWS Region, do the following in the **First Time Set Up** dialog box:

- a. Copy down the name of the Amazon S3 bucket that's shown. You'll need this information later.
 - b. Choose **Create S3 bucket** to let Amazon Rekognition Custom Labels create an Amazon S3 bucket (console bucket) on your behalf.
5. Close the browser window.

Step 6: (Optional) Encrypt training files

You can choose one of the following options to encrypt the Amazon Rekognition Custom Labels manifest files and image files that are in a console bucket or an external Amazon S3 bucket.

- Use an Amazon S3 key (SSE-S3).
- Use your AWS KMS key.

Note

The calling **IAM principal** need permissions to decrypt the files. For more information, see [Decrypting files encrypted with AWS Key Management Service \(p. 9\)](#).

For information about encrypting an Amazon S3 bucket, see [Setting default server-side encryption behavior for Amazon S3 buckets](#).

Decrypting files encrypted with AWS Key Management Service

If you use AWS Key Management Service (KMS) to encrypt your Amazon Rekognition Custom Labels manifest files and image files, add the IAM principal that calls Amazon Rekognition Custom Labels to the key policy of the KMS key. Doing this lets Amazon Rekognition Custom Labels decrypt your manifest and image files before training. For more information, see [My Amazon S3 bucket has default encryption using a custom AWS KMS key. How can I allow users to download from and upload to the bucket?](#)

The IAM principal needs the following permissions on the KMS key.

- kms:GenerateDataKey
- kms:Decrypt

For more information, see [Protecting Data Using Server-Side Encryption with KMS keys Stored in AWS Key Management Service \(SSE-KMS\)](#).

Encrypting copied training and test images

To train your model, Amazon Rekognition Custom Labels makes a copy of your source training and test images. By default the copied images are encrypted at rest with a key that AWS owns and manages. You can also choose to use your own AWS KMS key. If you use your own KMS key, you need the following permissions on the KMS key.

- kms>CreateGrant
- kms:DescribeKey

You optionally specify the KMS key when you train the model with the console or when you call the `CreateProjectVersion` operation. The KMS key you use doesn't need to be the same KMS key that you use to encrypt manifest and image files in your Amazon S3 bucket. For more information, see [Step 6: \(Optional\) Encrypt training files \(p. 9\)](#).

For more information, see [AWS Key Management Service concepts](#). Your source images are unaffected.

For information about training a model, see [Training an Amazon Rekognition Custom Labels model \(p. 106\)](#).

(Optional) Step 7: Associate prior datasets with new projects

Amazon Rekognition Custom Labels now manages datasets with projects. Earlier (prior) datasets that you created are read-only and must be associated with a project before you can use them. When you open the details page for a project with the console, we automatically associate the datasets that trained the latest version of the project's model with the project. Automatic association of a dataset with a project doesn't happen if you are using the AWS SDK.

Unassociated prior datasets have never been used to train a model or, were used to train a previous version of a model. The Prior datasets page shows all of your associated and unassociated datasets.

To use an unassociated prior dataset, you create a new project on the Prior datasets page. The dataset becomes the training dataset for the new project. You can also create a project for an already associated dataset as prior datasets can have multiple associations.

To associate a prior dataset to a new project

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. In the left pane, choose **Use Custom Labels**. The Amazon Rekognition Custom Labels landing page is shown.
3. In the left navigation pane, choose **Prior datasets**.
4. In the datasets view, choose the prior dataset that you want to associate with a project.
5. Choose **Create project with dataset**.
6. On the **Create project** page, enter a name for your new project in **Project name**.
7. Choose **Create project** to create the project. The project might take a while to create.
8. Use the project. For more information, see [Understanding Amazon Rekognition Custom Labels \(p. 27\)](#).

Using a prior dataset as a test dataset

You can use a prior dataset as the test dataset for an existing project by first associating the prior dataset with a new project. You then copy the training dataset of the new project to the test dataset of the existing project.

To use a prior dataset as a test dataset

1. Follow the instructions at [\(Optional\) Step 7: Associate prior datasets with new projects \(p. 10\)](#) to associate the prior dataset with a new project.
2. Create the test dataset in the existing project by using copying the training dataset from the new project. For more information, see [Existing dataset \(p. 64\)](#).
3. Follow the instructions at [Deleting an Amazon Rekognition Custom Labels project \(Console\) \(p. 200\)](#) to delete the new project.

Alternatively, you can create the test dataset by using the manifest file for prior dataset. For more information, see [Creating a manifest file \(p. 242\)](#).

Getting started with Amazon Rekognition Custom Labels

You use Amazon Rekognition Custom Labels to train a machine learning model. The trained model analyzes images to find the objects, scenes, and concepts that are unique to your business needs. For example, you can train a model to classify images of houses, or find the location of electronic parts on a printed circuit board.

To help you get started, Amazon Rekognition Custom Labels includes tutorial videos and example projects.

Topics

- [Tutorial videos \(p. 11\)](#)
- [Example projects \(p. 11\)](#)
- [Using the example projects \(p. 13\)](#)
- [Step 1: Choose an example project \(p. 14\)](#)
- [Step 2: Train your model \(p. 16\)](#)
- [Step 3: Start your model \(p. 18\)](#)
- [Step 4: Analyze an image with your model \(p. 19\)](#)
- [Step 5: Stop your model \(p. 24\)](#)
- [Step 6: Next steps \(p. 25\)](#)

Tutorial videos

The videos show you how to use Amazon Rekognition Custom Labels to train and use a model.

To view the tutorial videos

1. Sign in to the AWS Management Console and open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. In the left pane, choose **Use Custom Labels**. The Amazon Rekognition Custom Labels landing page is shown. If you don't see **Use Custom Labels**, check that the [AWS Region](#) you are using supports Amazon Rekognition Custom Labels.
3. In the navigation pane, choose **Get started**.
4. In **What is Amazon Rekognition Custom Labels?**, choose the video to watch the overview video.
5. In the navigation pane, choose **Tutorials**.
6. On the **Tutorials** page, choose the tutorial videos that you want to watch.

Example projects

Amazon Rekognition Custom Labels provides the following example projects.

Image classification

The image classification project (Rooms) trains a model that finds one or more household locations in an image, such as *backyard*, *kitchen*, and *patio*. The training and test images represent a single location. Each image is labeled with a single image-level label, such as *kitchen*, *patio*, or *living_space*. For an analyzed image, the trained model returns one or more matching labels from the set of image-level labels used for training. For example, the model might find the label *living_space* in the following image. For more information, see [Find objects, scenes, and concepts \(p. 55\)](#).



Multi-label image classification

The multi-label image classification project (Flowers) trains a model that categorizes images of flowers into three concepts (flower type, leaf presence, and growth stage).

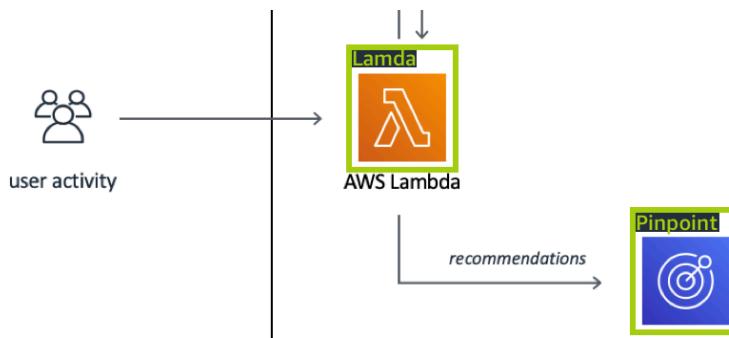
The training and test images have image-level labels for each concept, such as *camellia* for a flower type, *with_leaves* for a flower with leaves, and *fully_grown* for a flower that is fully grown.

For an analyzed image, the trained model returns matching labels from the set of image-level labels used for training. For example, the model returns the labels *mediterranean_spurge* and *with_leaves* for the following image. For more information, see [Find objects, scenes, and concepts \(p. 55\)](#).



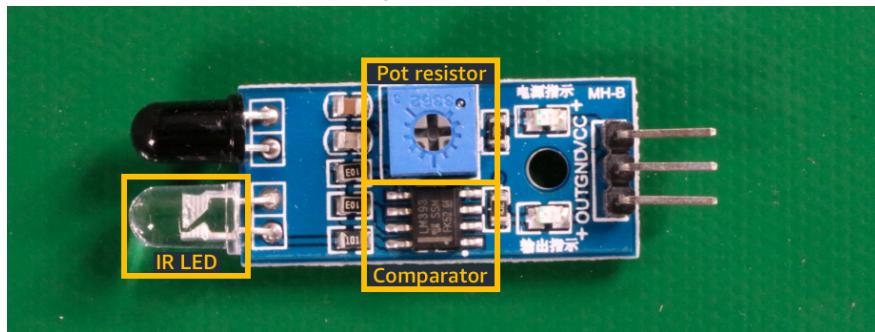
Brand detection

The brand detection project (Logos) trains a model that finds the location of certain AWS logos such as *Amazon Textract*, and *AWS lambda*. The training images are of the logo only and have a single image level-label, such as *lambda* or *textract*. It is also possible to train a brand detection model with training images that have bounding boxes for brand locations. The test images have labeled bounding boxes that represent the location of logos in natural locations, such as an architectural diagram. The trained model finds the logos and returns a labeled bounding box for each logo found. For more information, see [Find brand locations \(p. 56\)](#).



Object localization

The object localization project (Circuit boards) trains a model that finds the location of parts on a printed circuit board, such as a *comparator* or an *infra red light emitting diode*. The training and test images include bounding boxes that surround the circuit board parts and a label that identifies the part within the bounding box. The label names are *ir_phototransistor*, *ir_led*, *pot_resistor*, and *comparator*. The trained model finds the circuit board parts and returns a labeled bounding for each circuit part found. For more information, see [Find object locations \(p. 56\)](#).



Using the example projects

These Getting Started instructions show you how to train a model by using example projects that Amazon Rekognition Custom Labels creates for you. It also shows you how to start the model and use it to analyze an image.

Creating the example project

To get started, decide which project to use. For more information, see [Step 1: Choose an example project \(p. 14\)](#).

Amazon Rekognition Custom Labels uses datasets to train and evaluate (test) a model. A dataset manages images and the labels that identify the contents of images. The example projects include a training dataset and a test dataset in which all images are labeled. You don't need to make any changes before training your model. The example projects show the two ways in which Amazon Rekognition Custom Labels uses labels to train different types of models.

- *image-level* – The label identifies an object, scene, or concept that represents the entire image.
- *bounding box* – The label identifies the contents of a bounding box. A bounding box is a set of image coordinates that surround an object in an image.

Later, when you create a project with your own images, you must create training and test datasets, and also label your images. For more information, see [Decide your model type \(p. 27\)](#).

Training the model

After Amazon Rekognition Custom Labels creates the example project, you can train the model. For more information, see [Step 2: Train your model \(p. 16\)](#). After training finishes, you normally evaluate the performance of the model. The images in the example dataset already create a high-performance model, and you don't need to evaluate the model before running the model. For more information, see [Improving a trained Amazon Rekognition Custom Labels model \(p. 152\)](#).

Using the model

Next you start the model. For more information, see [Step 3: Start your model \(p. 18\)](#).

After you start running your model, you can use it to analyze new images. For more information, see [Step 4: Analyze an image with your model \(p. 19\)](#).

You are charged for the amount of time that your model runs. When you finish using the example model, you should stop the model. For more information, see [Step 5: Stop your model \(p. 24\)](#).

Next steps

When you're ready, you can create your own projects. For more information, see [Step 6: Next steps \(p. 25\)](#).

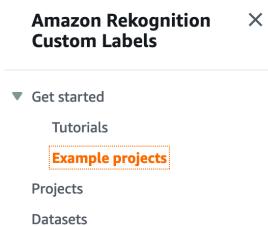
Step 1: Choose an example project

In this step you use choose an example project. Amazon Rekognition Custom Labels then creates a project and a dataset for you. A project manages the files used to train your model. For more information, see [Managing an Amazon Rekognition Custom Labels project \(p. 200\)](#). Datasets contain the images, assigned labels, and bounding boxes that you use to train and test a model. For more information, see [the section called "Managing datasets" \(p. 211\)](#).

For information about the example projects, see [Example projects \(p. 11\)](#).

Choose an example project

1. Sign in to the AWS Management Console and open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. In the left pane, choose **Use Custom Labels**. The Amazon Rekognition Custom Labels landing page is shown. If you don't see **Use Custom Labels**, check that the **AWS Region** you are using supports Amazon Rekognition Custom Labels.
3. Choose **Get started**.

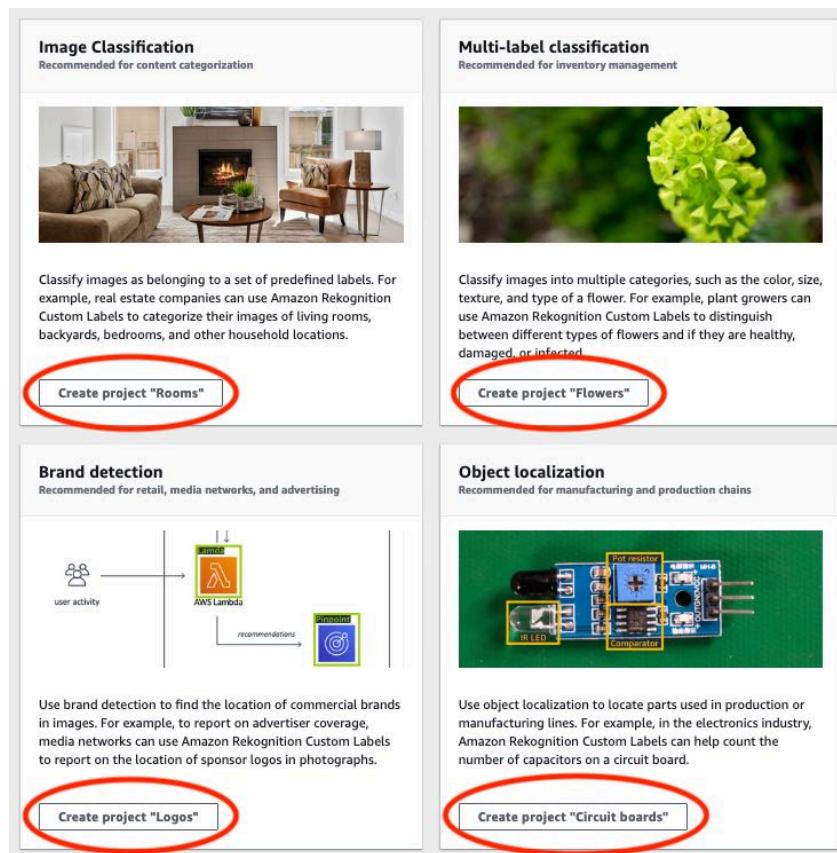


4. In **Explore example projects**, choose **Try example projects**.
5. Decide which project you want to use and choose **Create project "project name"** within the example section. Amazon Rekognition Custom Labels then creates the example project for you.

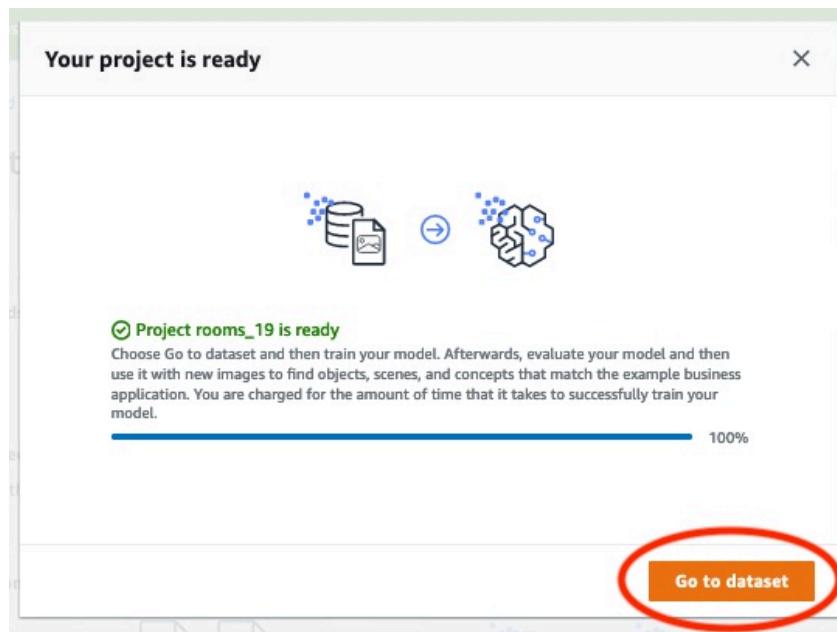
Note

If this is the first time that you've opened the console in the current AWS Region, the **First Time Set Up** dialog box is shown. Do the following:

1. Note the name of the Amazon S3 bucket that's shown.
2. Choose **Continue** to let Amazon Rekognition Custom Labels create an Amazon S3 bucket (console bucket) on your behalf.



6. After your project is ready, choose **Go to dataset**.

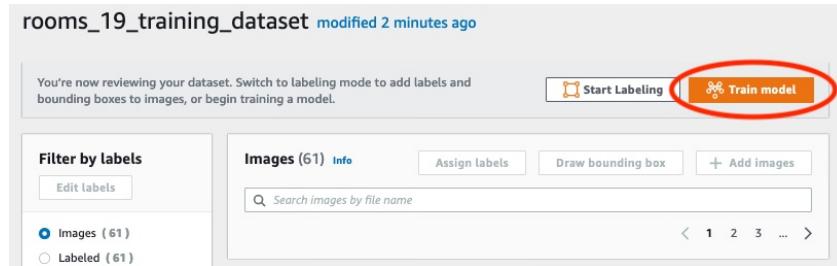


Step 2: Train your model

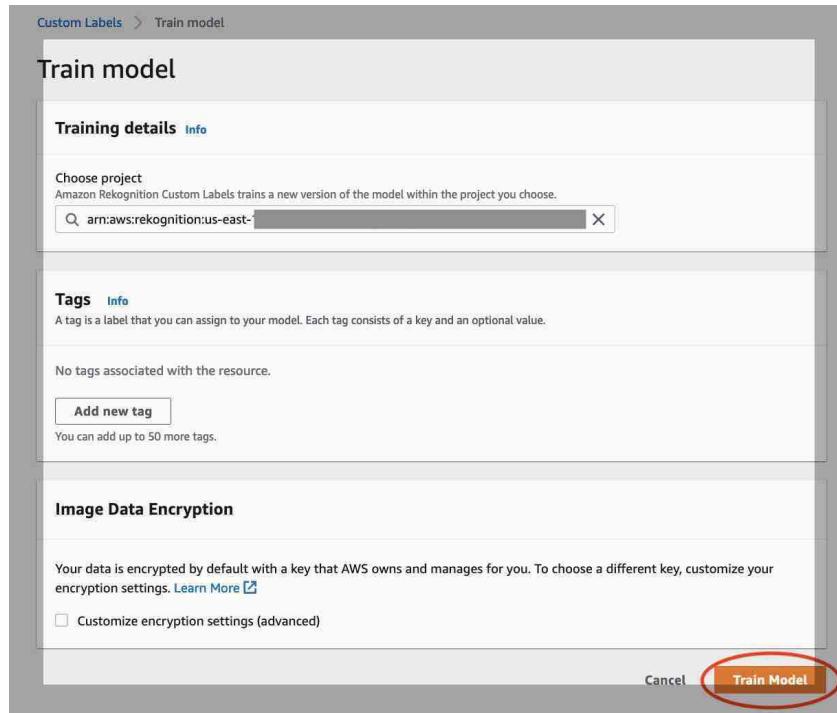
In this step you train your model. The training and test datasets are automatically configured for you. After training successfully completes, you can see the overall evaluation results, and evaluation results for individual test images. For more information, see [Training an Amazon Rekognition Custom Labels model \(p. 106\)](#).

To train your model

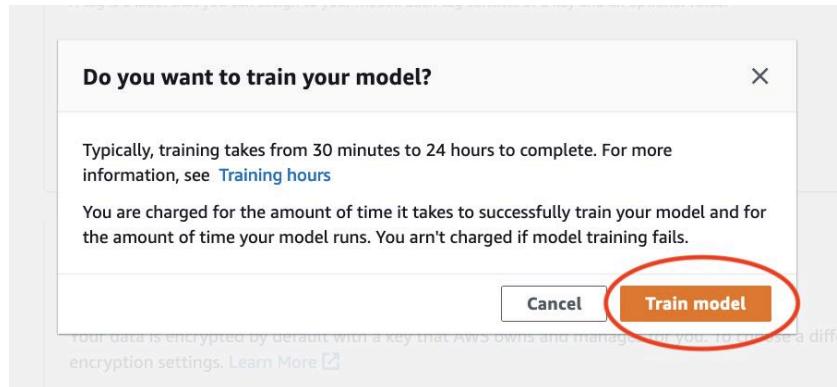
1. On the dataset page, choose **Train model**.



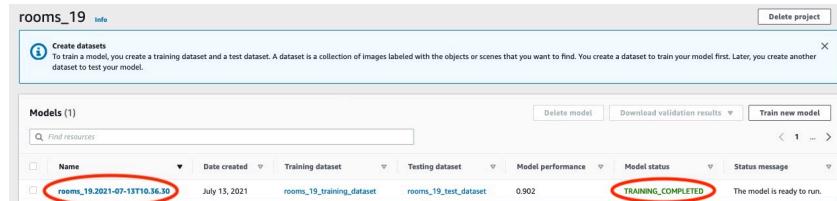
2. On the **Train model** page, Choose **Train model**. The Amazon Resource Name (ARN) for your project is in the **Choose project** edit box.



3. In the **Do you want to train your model?** dialog box, choose **Train model**.



4. After training completes, choose the model name. Training is finished when the model status is **TRAINING_COMPLETED**.



5. Choose the **Evaluate** button to see the evaluation results. For information about evaluating a model, see [Improving a trained Amazon Rekognition Custom Labels model \(p. 152\)](#).
6. Choose **View test results** to see the results for individual test images.

Evaluation results

F1 score <small>Info</small>	Average precision <small>Info</small>	Overall recall <small>Info</small>
0.902	0.893	0.928
Date completed	Training dataset	Testing dataset
July 13, 2021 Trained in 1.223 hours	10 labels, 61 images	10 labels, 56 images

Per label performance (10)

Label name	F1 score	Test images	Precision	Recall	Assumed threshold
backyard	0.857	4	1.000	0.750	0.286
bathroom	0.889	9	0.889	0.889	0.185
bedroom	0.900	11	1.000	0.818	0.262
closet	1.000	2	1.000	1.000	0.169
entry_way	1.000	3	1.000	1.000	0.149
floor_plan	1.000	2	1.000	1.000	0.685

7. After viewing the test results, choose the project name to return to the model page.

Evaluate image
Review the test results of your trained model for individual images. Below each image is information about the model's predicted label compared with the label assigned to the image in the test dataset, noted by result type. You can also filter by label and result types.

Images (56) <small>Info</small>		
backyard2.jpeg	Labels front_yard backyard	Confidence 30.3% 21.6%
backyard4.jpeg	Labels backyard	Confidence 46.3%

Step 3: Start your model

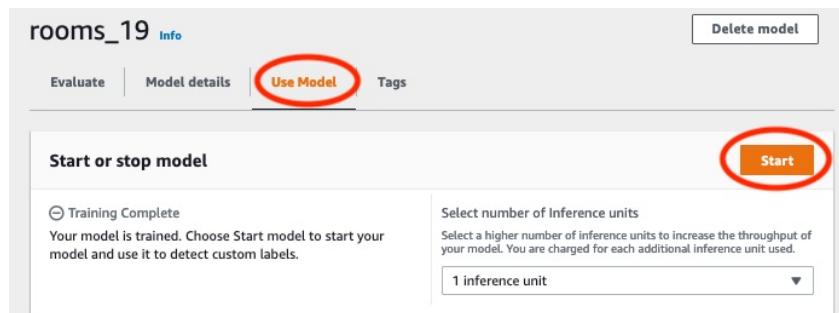
In this step you start your model. After your model starts, you can use it to analyze images.

You are charged for the amount of time that your model runs. Stop your model if you don't need to analyze images. You can restart your model at a later time. For more information, see [Running a trained Amazon Rekognition Custom Labels model \(p. 165\)](#).

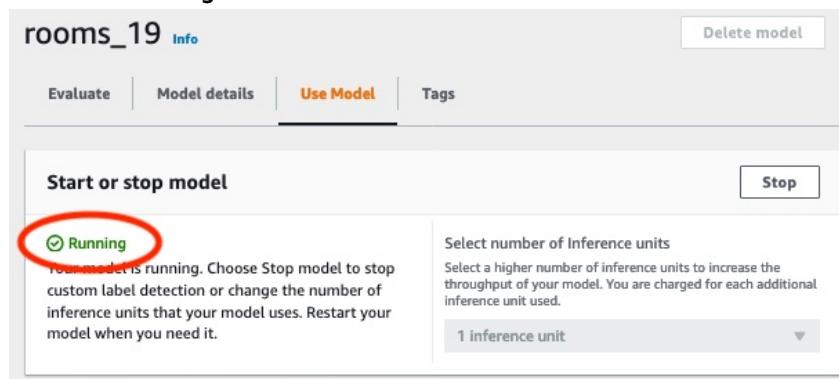
To start your model

1. Choose the **Use model** tab on the model page.

2. In the **Start or stop model** section do the following:
 - a. Choose **Start**.
 - b. In the **Start model** dialog box, choose **Start**.



3. Wait until the model is running. The model is running when the status in the **Start or stop model** section is **Running**.



4. Use your model to classify images. For more information, see [Step 4: Analyze an image with your model \(p. 19\)](#).

Step 4: Analyze an image with your model

You analyze an image by calling the `DetectCustomLabels` API. In this step, you use the `detect-custom-labels` AWS Command Line Interface (AWS CLI) command to analyze an example image. You get the AWS CLI command from the Amazon Rekognition Custom Labels console. The console configures the AWS CLI command to use your model. You only need to supply an image that's stored in an Amazon S3 bucket. This topic provides an image that you can use for each example project.

Note

The console also provides Python example code.

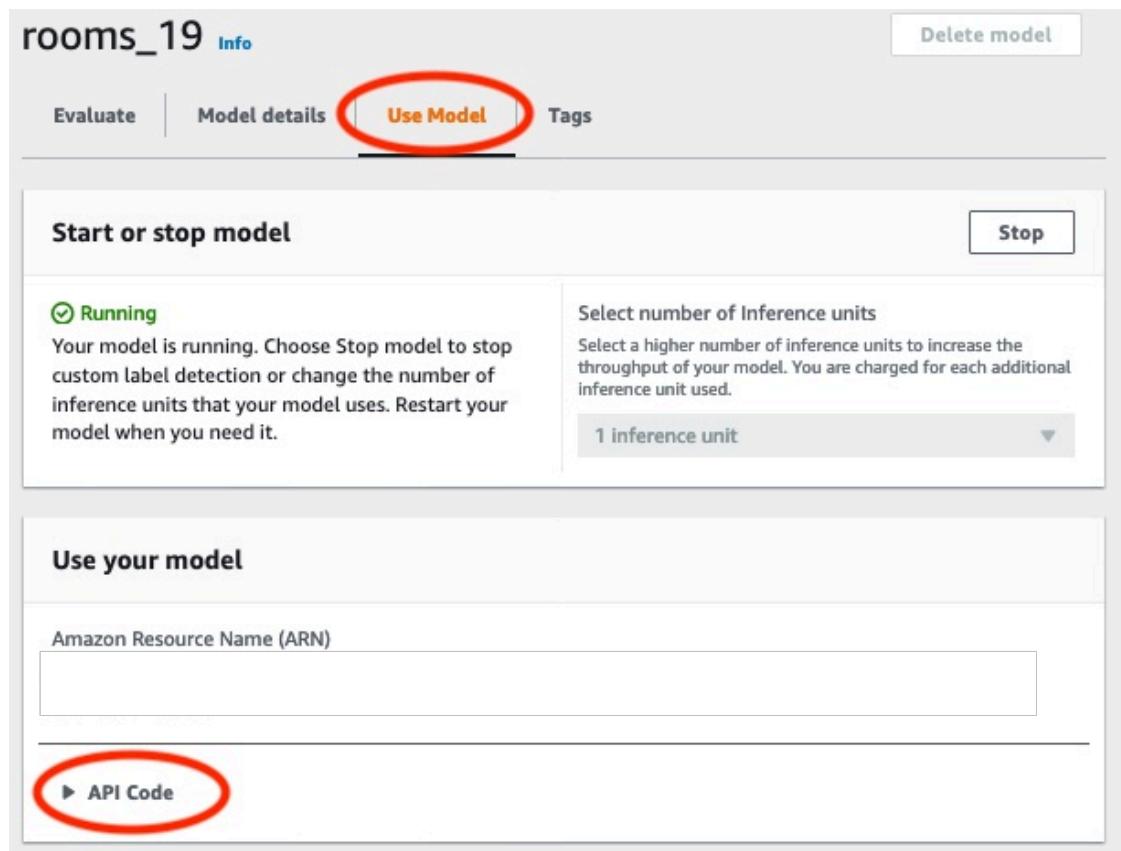
The output from `detect-custom-labels` includes a list of labels found in the image, bounding boxes (if the model finds object locations), and the confidence that the model has in the accuracy of the predictions.

For more information, see [Analyzing an image with a trained model \(p. 182\)](#).

To analyze an image (console)

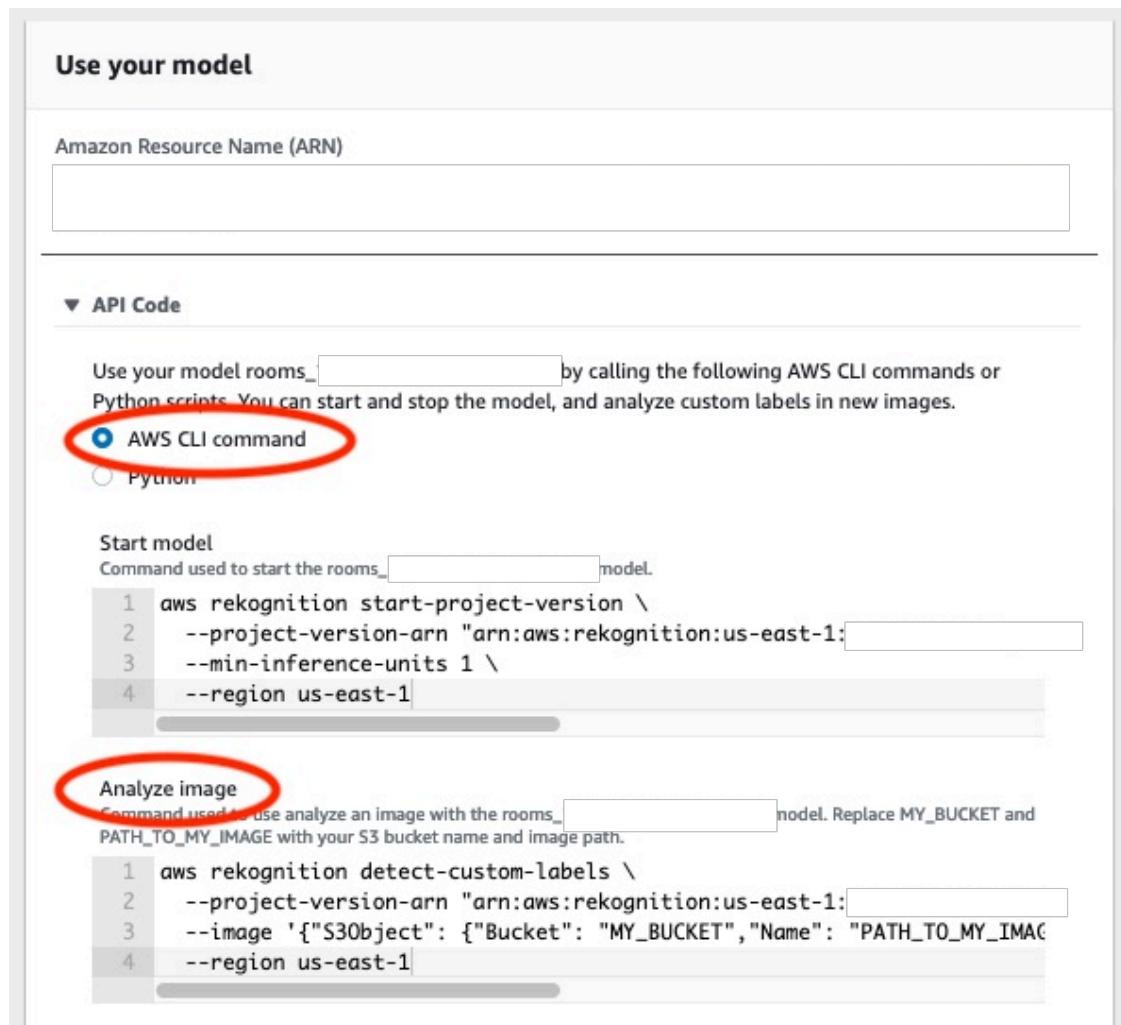
1. If you haven't already, set up the AWS CLI. For instructions, see [the section called "Step 3: Set Up the AWS CLI and AWS SDKs" \(p. 5\)](#).

2. If you haven't already, start running your model. For more information, see [Step 2: Train your model \(p. 16\)](#).
3. Choose the **Use Model** tab and then choose **API code**.



The screenshot shows the 'Use Model' tab selected in the navigation bar. The 'API Code' button is highlighted with a red oval. The 'Start or stop model' section shows the model is 'Running'. The 'Use your model' section contains an 'Amazon Resource Name (ARN)' input field and a '▶ API Code' button, which is also highlighted with a red oval.

4. Choose **AWS CLI command**.
5. In the **Analyze image** section, copy the AWS CLI command that calls `detect-custom-labels`.



Use your model

Amazon Resource Name (ARN)

▼ API Code

Use your model rooms by calling the following AWS CLI commands or Python scripts. You can start and stop the model, and analyze custom labels in new images.

AWS CLI command

Python

Start model

Command used to start the rooms model.

```
1 awsrekognition start-project-version \
2   --project-version-arn "arn:aws:rekognition:us-east-1:12345678901234567890" \
3   --min-inference-units 1 \
4   --region us-east-1
```

Analyze image

Command used to use analyze an image with the rooms model. Replace MY_BUCKET and PATH_TO_MY_IMAGE with your S3 bucket name and image path.

```
1 awsrekognition detect-custom-labels \
2   --project-version-arn "arn:aws:rekognition:us-east-1:12345678901234567890" \
3   --image '{"S3Object": {"Bucket": "MY_BUCKET", "Name": "PATH_TO_MY_IMAGE"}' \
4   --region us-east-1
```

6. Upload an example image to an Amazon S3 bucket. For instructions, see [Getting an example image \(p. 22\)](#).
7. At the command prompt, enter the AWS CLI command that you copied in the previous step. It should look like the following example.

The value of `--project-version-arn` should be Amazon Resource Name (ARN) of your model. The value of `--region` should be the AWS Region in which you created the model.

Change `MY_BUCKET` and `PATH_TO_MY_IMAGE` to the Amazon S3 bucket and image that you used in the previous step.

```
awsrekognition detect-custom-labels \
  --project-version-arn "model_arn" \
  --image '{"S3Object": {"Bucket": "MY_BUCKET", "Name": "PATH_TO_MY_IMAGE"}' \
  --region us-east-1
```

If the model finds objects, scenes, and concepts, the JSON output from the AWS CLI command should look similar to the following. `Name` is the name of the image-level label that the model found. `Confidence` (0-100) is the model's confidence in the accuracy of the prediction.

```
{
```

```
"CustomLabels": [  
  {  
    "Name": "living_space",  
    "Confidence": 83.41299819946289  
  }  
]
```

If the model finds object locations or finds brand, labeled bounding boxes are returned. BoundingBox contains the location of a box that surrounds the object. Name is the object that the model found in the bounding box. Confidence is the model's confidence that the bounding box contains the object.

```
{  
  "CustomLabels": [  
    {  
      "Name": "textract",  
      "Confidence": 87.7729721069336,  
      "Geometry": {  
        "BoundingBox": {  
          "Width": 0.198987677693367,  
          "Height": 0.31296101212501526,  
          "Left": 0.07924537360668182,  
          "Top": 0.4037395715713501  
        }  
      }  
    }  
  ]  
}
```

8. Continue to use the model to analyze other images. Stop the model if you are no longer using it. For more information, see [Step 5: Stop your model \(p. 24\)](#).

Getting an example image

You can use the following images with the `DetectCustomLabels` operation. There is one image for each project. To use the images, you upload them to an S3 bucket.

To use an example image

1. Right-click the following image that matches the example project that you are using. Then choose **Save image** to save the image to your computer. The menu option might be different, depending on which browser you are using.
2. Upload the image to an Amazon S3 bucket that's owned by your AWS account and is in the same AWS region in which you are using Amazon Rekognition Custom Labels.

For instructions, see [Uploading Objects into Amazon S3](#) in the *Amazon Simple Storage Service User Guide*.

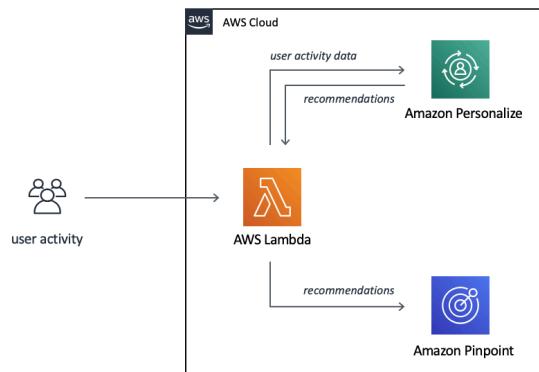
Image classification



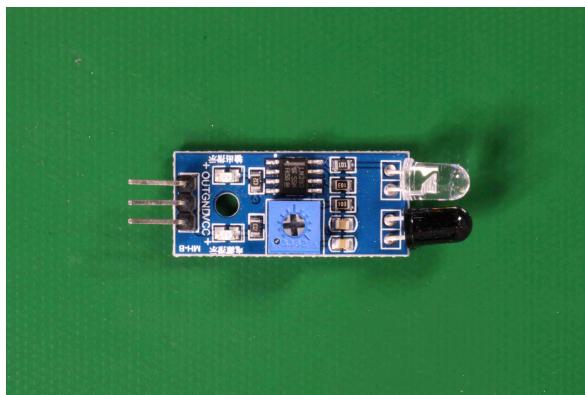
Multi-label classification



Brand detection



Object localization

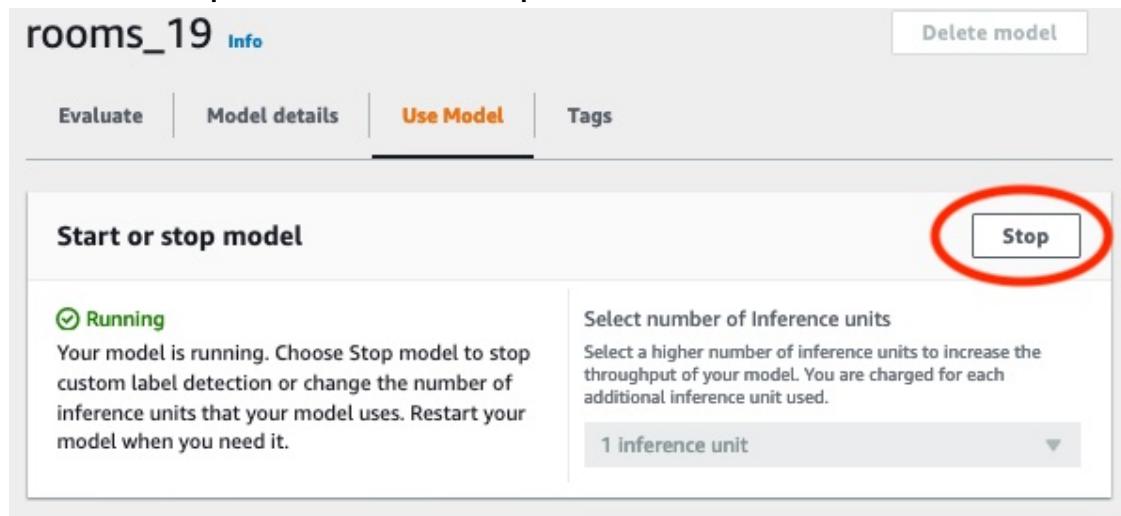


Step 5: Stop your model

In this step you stop running your model. You are charged for the amount of time your model is running. If you have finished using the model, you should stop it.

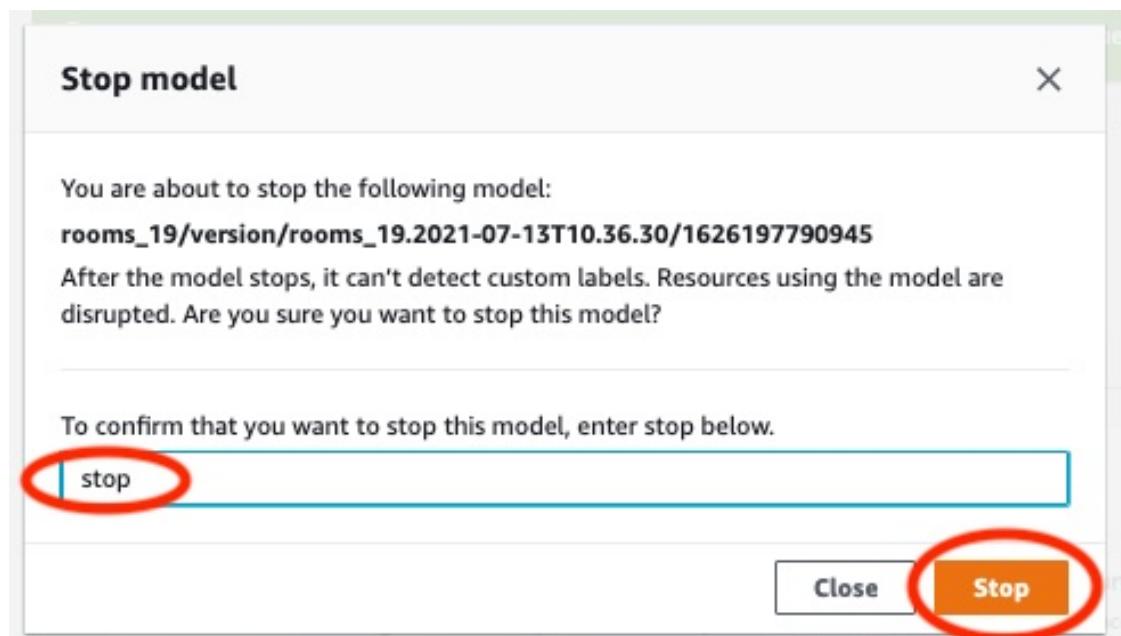
To stop your model

1. In the **Start or stop model** section choose **Stop**.

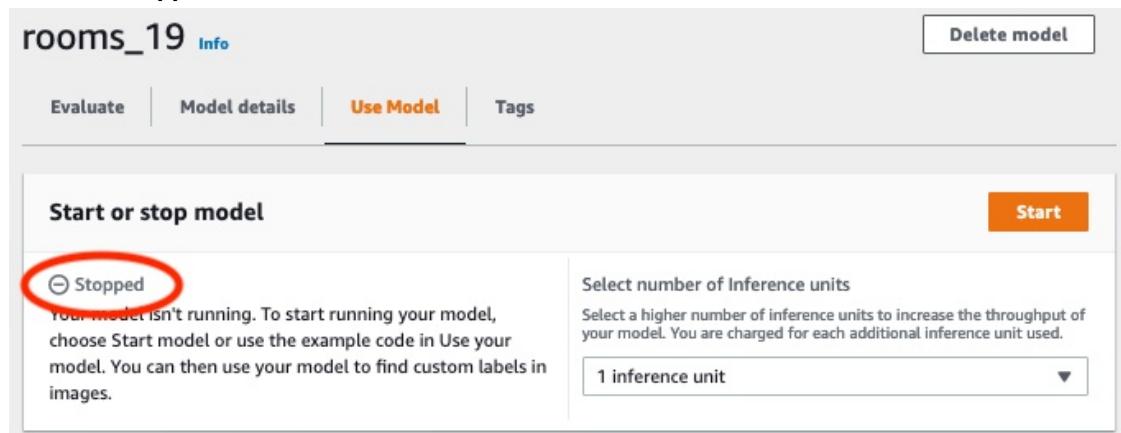


The screenshot shows the 'Use Model' section of the Rekognition Custom Labels interface. At the top, the model name 'rooms_19' is displayed with an 'Info' link and a 'Delete model' button. Below this, there are tabs for 'Evaluate', 'Model details', 'Use Model' (which is selected and highlighted in orange), and 'Tags'. The 'Use Model' section contains a 'Start or stop model' button. To the left of the button, there is a status message: 'Running' with a green checkmark, followed by the text: 'Your model is running. Choose Stop model to stop custom label detection or change the number of inference units that your model uses. Restart your model when you need it.' To the right of the button, there is a 'Select number of Inference units' section with a dropdown menu set to '1 inference unit'. The 'Stop' button is highlighted with a red circle.

2. In the **Stop model** dialog box, enter **stop** to confirm that you want to stop the model.



3. Choose **Stop** to stop your model. The model has stopped when the status in the **Start or stop model** section is **Stopped**.



Step 6: Next steps

After you finished trying the examples projects, you can use your own images and datasets to create your own model. For more information, see [Understanding Amazon Rekognition Custom Labels \(p. 27\)](#).

Use the labeling information in the following table to train models similar to the example projects.

Example	Training images	Test images
Image classification (Rooms)	1 Image-level label per image	1 Image-level label per image
Multi-label classification (Flowers)	Multiple image-level labels per image	Multiple image-level labels per image

Example	Training images	Test images
Brand detection (Logos)	image level-labels (you can also use Labeled bounding boxes)	Labeled bounding boxes
Image localization (Circuit boards)	Labeled bounding boxes	Labeled bounding boxes

The [Tutorial: Classifying images \(p. 34\)](#) shows you how to create a project, datasets, and models for an Image classification model.

For detailed information about creating datasets and training models, see [Creating an Amazon Rekognition Custom Labels model \(p. 50\)](#).

Understanding Amazon Rekognition Custom Labels

This section gives you an overview of the workflow to train and use an Amazon Rekognition Custom Labels model with the console and the AWS SDK.

Note

Amazon Rekognition Custom Labels now manages datasets within a project. You can create datasets for your projects with the console and with the AWS SDK. If you have previously used Amazon Rekognition Custom Labels, your older datasets might need associating with a new project. For more information, see [\(Optional\) Step 7: Associate prior datasets with new projects \(p. 10\)](#)

Topics

- [Decide your model type \(p. 27\)](#)
- [Create a model \(p. 29\)](#)
- [Improve your model \(p. 31\)](#)
- [Start your model \(p. 32\)](#)
- [Analyze an image \(p. 32\)](#)
- [Stop your model \(p. 33\)](#)

Decide your model type

You first decide which type of model you want to train, which depends on your business goals. For example, you could train a model to find your logo in social media posts, identify your products on store shelves, or classify machine parts in an assembly line.

Amazon Rekognition Custom Labels can train the following types of model:

- [Find objects, scenes, and concepts \(p. 27\)](#)
- [Find object locations \(p. 28\)](#)
- [Find the location of brands \(p. 29\)](#)

To help you decide which type of model to train, Amazon Rekognition Custom Labels provides example projects that you can use. For more information, see [Getting started with Amazon Rekognition Custom Labels \(p. 11\)](#).

Find objects, scenes, and concepts

The model predicts objects, scenes, and concepts associated with an entire image. For example, you can train a model that determines if an image contains a *tourist attraction*, or not.

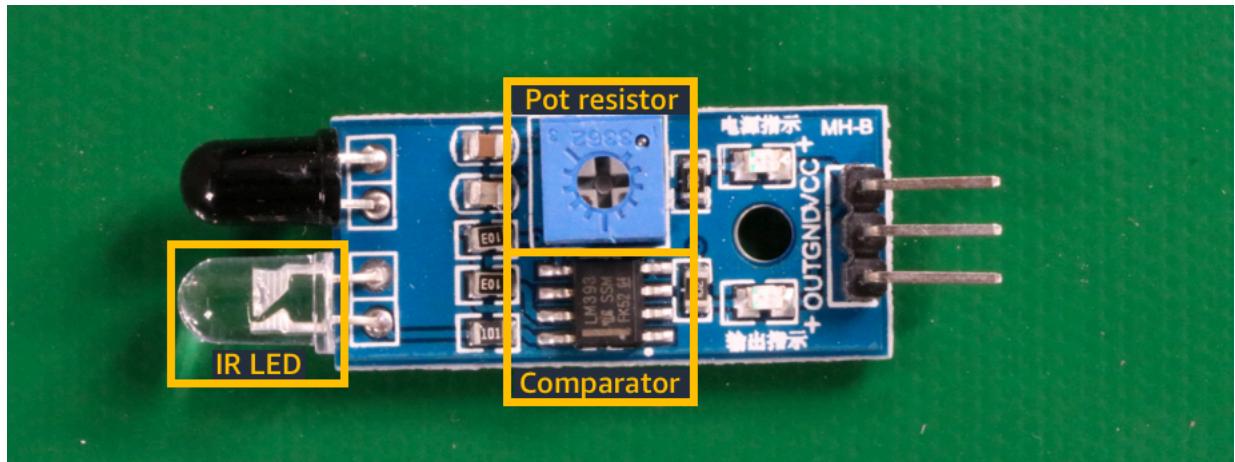


Alternatively, you can train a model that categorizes images into multiple categories. For example, the previous image might have categories such as *sky color*, *reflection*, or *lake*.

The [Image classification \(p. 12\)](#) and [Multi-label image classification \(p. 12\)](#) example projects show different uses for image-level labels.

Find object locations

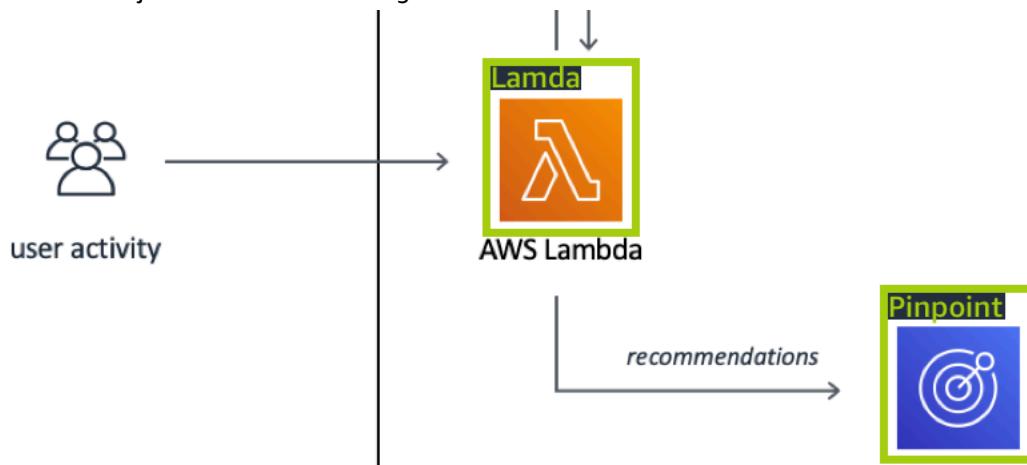
The model predicts the location of an object on an image. The prediction includes bounding box information for the object location and a label that identifies the object within the bounding box. For example, the following image shows bounding boxes around various parts of a circuit board, such as a *comparator* or *pot resistor*.



The [Object localization \(p. 13\)](#) example project shows how Amazon Rekognition Custom Labels uses labeled bounding boxes to train a model that finds object locations.

Find the location of brands

Amazon Rekognition Custom Labels can train a model that finds the location of brands, such as logos, on an image. The prediction includes bounding box information for the brand location and a label that identifies the object within the bounding box.



Create a model

The steps to create a model are creating a project, creating training and test datasets, and training the model.

Create a project

An Amazon Rekognition Custom Labels project is a group of resources needed to create and manage a model. A project manages the following:

- **Datasets** – The images and image labels used to train a model. A project has a training dataset and a test dataset.

- **Models** – The software that you train to find the concepts, scenes, and objects unique to your business. You can have multiple versions of a model in a project.

We recommend that you use a project for a single use case, such as finding circuit board parts on a circuit board.

You can create a project with the Amazon Rekognition Custom Labels console and with the [CreateProject API](#). For more information, see [Creating a project \(p. 50\)](#).

Create training and test datasets

A dataset is a set of images and labels that describe those images. Within your project, you create a training dataset and a test dataset that Amazon Rekognition Custom Labels uses to train and test your model.

A label identifies an object, scene, concept, or bounding box around an object in an image. Labels are either assigned to an entire image (*image-level*) or they are assigned to a bounding box that surrounds an object on an image.

Important

How you label the images in your datasets determines the type of model that Amazon Rekognition Custom Labels creates. For example, to train a model that finds objects, scenes and concepts, you assign image level labels to the images in your training and test datasets. For more information, see [Purposing datasets \(p. 55\)](#).

Images must be in PNG and JPEG format, and you should follow the input images recommendations. For more information, see [Preparing images \(p. 58\)](#).

Create training and test datasets (Console)

You can start a project with a single dataset, or with separate training and test datasets. If you start with a single dataset, Amazon Rekognition Custom Labels splits your dataset during training to create a training dataset (80%) and a test dataset (20%) for your project. Start with a single dataset if you want Amazon Rekognition Custom Labels to decide which images are used for training and testing. For complete control over training, testing, and performance tuning, we recommend that you start your project with separate training and test datasets.

To create the datasets for a project, you import the images in one of the following ways:

- Import images from your local computer.
- Import images from an S3 bucket. Amazon Rekognition Custom Labels can label the images using the folder names that contain the images.
- Import an Amazon SageMaker Ground Truth manifest file.
- Copy an existing Amazon Rekognition Custom Labels dataset.

For more information, see [Creating training and test datasets \(Console\) \(p. 59\)](#).

Depending on where you import your images from, your images might be unlabeled. For example, images imported from a local computer aren't labeled. Images imported from an Amazon SageMaker Ground Truth manifest file are labeled. You can use the Amazon Rekognition Custom Labels console to add, change, and assign labels. For more information, see [Labeling images \(p. 99\)](#).

To create your training and test datasets with the console, see [Creating training and test datasets \(Console\) \(p. 59\)](#). For a tutorial that includes creating training and test datasets, see [Tutorial: Classifying images \(p. 34\)](#).

Create training and test datasets (SDK)

To create your training and test datasets, you use the [CreateDataset API](#). You can create a dataset by using an Amazon Sagemaker format manifest file or by copying an existing Amazon Rekognition Custom Labels dataset. For more information, see [Create training and test datasets \(SDK\) \(p. 65\)](#) If necessary, you can create your own manifest file. For more information, see the section called “[Creating a manifest file](#)” (p. 242).

Train your model

Train your model with the training dataset. A new version of a model is created each time it is trained. During training, Amazon Rekognition Custom Labels test the performance of your trained model. You can use the results to evaluate and improve your model. Training takes a while to complete. You are only charged for a successful model training. For more information, see [Training an Amazon Rekognition Custom Labels model \(p. 106\)](#). If model training fails, Amazon Rekognition Custom Labels provides debugging information that you can use. For more information, see [Debugging a failed model training \(p. 116\)](#).

Train your model (Console)

To train your model with the console, see [Training a model \(Console\) \(p. 107\)](#).

Training a model (SDK)

You train an Amazon Rekognition Custom Labels model by calling [CreateProjectVersion](#).

Improve your model

During testing, Amazon Rekognition Custom Labels creates evaluation metrics that you can use to improve your trained model.

Evaluate your model

Evaluate the performance of your model by using the performance metrics created during testing. Performance metrics, such as F1, precision, and recall, allow you to understand the performance of your trained model, and decide if you're ready to use it in production. For more information, see [Metrics for evaluating your model \(p. 152\)](#).

Evaluate a model (console)

To view performance metrics, see [Accessing evaluation metrics \(Console\) \(p. 155\)](#).

Evaluate a model (SDK)

To get performance metrics, you call [DescribeProjectVersions](#) to get the testing results. For more information, see [Accessing Amazon Rekognition Custom Labels evaluation metrics \(SDK\) \(p. 156\)](#). The testing results include metrics not available in the console, such as a confusion matrix for classification results. The testing results are returned in the following formats:

- F1 score – A single value representing the overall performance of precision and recall for the model. For more information, see [F1 \(p. 154\)](#).
- Summary file location – The testing summary includes aggregated evaluation metrics for the entire testing dataset and metrics for each individual label. [DescribeProjectVersions](#) returns the S3 bucket and folder location of the summary file. For more information, see [Summary file \(p. 157\)](#).

- Evaluation manifest snapshot location – The snapshot contains details about the test results, including the confidence ratings and the results of binary classification tests, such as false positives. `DescribeProjectVersions` returns the S3 bucket and folder location of the snapshot files. For more information, see [Evaluation manifest snapshot \(p. 158\)](#).

Improve your model

If improvements are needed, you can add more training images or improve dataset labeling. For more information, see [Improving an Amazon Rekognition Custom Labels model \(p. 163\)](#). You can also give feedback on the predictions your model makes and use it to make improvements to your model. For more information, see [Model feedback solution \(p. 309\)](#).

Improve your model (console)

To add images to a dataset, see [Adding more images to a dataset \(p. 218\)](#). To add or change labels, see [the section called “Labeling images” \(p. 99\)](#).

To retrain your model, see [Training a model \(Console\) \(p. 107\)](#).

Improve your model (SDK)

To add images to a dataset or change the labeling for an image, use the `UpdateDatasetEntries` API. `UpdateDatasetEntries` updates or adds JSON lines to a manifest file. Each JSON line contains information for a single image, such as assigned labels or bounding box information. For more information, see [Adding more images \(SDK\) \(p. 218\)](#). To view the entries in a dataset, use the `ListDatasetEntries` API.

To retrain your model, see [Training a model \(SDK\) \(p. 110\)](#).

Start your model

Before you can use your model, you start the model by using the Amazon Rekognition Custom Labels console or the `StartProjectVersion` API. You are charged for the amount of time that your model runs. For more information, see [Running a trained model \(p. 165\)](#).

Start your model (console)

To start your model using the console, see [Starting an Amazon Rekognition Custom Labels model \(Console\) \(p. 168\)](#).

Start your model

You start your model calling `StartProjectVersion`. For more information, see [Starting an Amazon Rekognition Custom Labels model \(SDK\) \(p. 168\)](#).

Analyze an image

To analyze an image with your model, you use the `DetectCustomLabels` API. You can specify a local image, or an image stored in an S3 bucket. The operation also requires the Amazon Resource Name (ARN) of the model that you want to use.

If your model finds objects, scenes, and concepts, the response includes a list of image-level labels found in the image. For example, the following image shows the image-level labels found using *Rooms* example project.



If the model finds object locations, the response includes list of labeled bounding boxes found in the image. A bounding box represents the location of an object on an image. You can use the bounding box information to draw a bounding box around an object. For example, the following image shows bounding boxes around circuit board parts found using the *Circuit boards* example project.



For more information, see [Analyzing an image with a trained model \(p. 182\)](#).

Stop your model

You are charged for the time that your model is running. If you are no longer using your model, stop the model by using the Amazon Rekognition Custom Labels console, or by using the `StopProjectVersion` API. For more information, see [Stopping an Amazon Rekognition Custom Labels model \(p. 175\)](#).

Stop your model (SDK)

To stop a running model with the console, see [Stopping an Amazon Rekognition Custom Labels model \(Console\) \(p. 175\)](#).

Stop your model (SDK)

To stop a running model, call `StopProjectVersion`. For more information, see [Stopping an Amazon Rekognition Custom Labels model \(SDK\) \(p. 176\)](#).

Tutorial: Classifying images

This tutorial shows you how to create the project and datasets for a model that classifies objects, scenes, and concepts found in an image. The model classifies the entire image. For example, by following this tutorial, you can train a model to recognize household locations such as a living room or kitchen. The tutorial also shows you how to use the model to analyze images.

Before starting the tutorial, we recommend that you read [Understanding Amazon Rekognition Custom Labels \(p. 27\)](#).

In this tutorial, you create the training and test datasets by uploading images from your local computer. Later you assign image-level labels to the images in your training and test datasets.

The model you create classifies images as belonging to the set of image-level labels that you assign to the training dataset images. For example, if the set of image-level labels in your training dataset is kitchen, living_room, patio, and backyard, the model can potentially find all of those image-level labels in a single image.

Note

You can create models for different purposes such as finding the location of objects on an image. For more information, see [Decide your model type \(p. 27\)](#).

Step 1: Collect your images

You need two sets of images. One set to add to your training dataset. Another set to add to your test dataset. The images should represent the objects, scenes, and concepts that you want your model to classify. The images must be in PNG or JPEG format. For more information, see [Preparing images \(p. 58\)](#).

You should have at least 10 images for your training dataset and 10 images for your test dataset.

If you don't yet have images, use the images from the *Rooms* example classification project. After creating the project, the training and test images are at the following Amazon S3 bucket locations:

- Training images — `s3://custom-labels-console-region-numbers/assets/rooms_version number_test_dataset/`
- Test images — `s3://custom-labels-console-region-numbers/assets/rooms_version number_test_dataset/`

region is the AWS Region in which you are using the Amazon Rekognition Custom Labels console. *numbers* is a value that the console assigns to the bucket name. *Version number* is the version number for the example project, starting at 1.

The following procedure stores images from the *Rooms* project into local folders on your computer named *training* and *test*.

To download the *Rooms* example project image files

1. Create the *Rooms* project. For more information, see [Step 1: Choose an example project \(p. 14\)](#).

2. Open the command prompt and enter the following command to download the training images.

```
aws s3 cp s3://custom-labels-console-region-numbers/assets/rooms_version number_training_dataset/ training --recursive
```

3. At the command prompt, enter the following command to download the test images.

```
aws s3 cp s3://custom-labels-console-region-numbers/assets/rooms_version number_test_dataset/ test --recursive
```

4. Move two of the images from the training folder to a separate folder of your choosing. You'll use the images to try your trained model in [Step 9: Analyze an image with your model \(p. 45\)](#).

Step 2: Decide your classes

Make a list of the classes that you want your model to find. For example, if you're training a model to recognize rooms in a house, you can classify the following image as `living_room`.



Each class maps to an image-level label. Later you assign image-level labels to the images in your training and test datasets.

If you're using the images from the Rooms example project, the image-level labels are `backyard`, `bathroom`, `bedroom`, `closet`, `entry_way`, `floor_plan`, `front_yard`, `kitchen`, `living_space`, and `patio`.

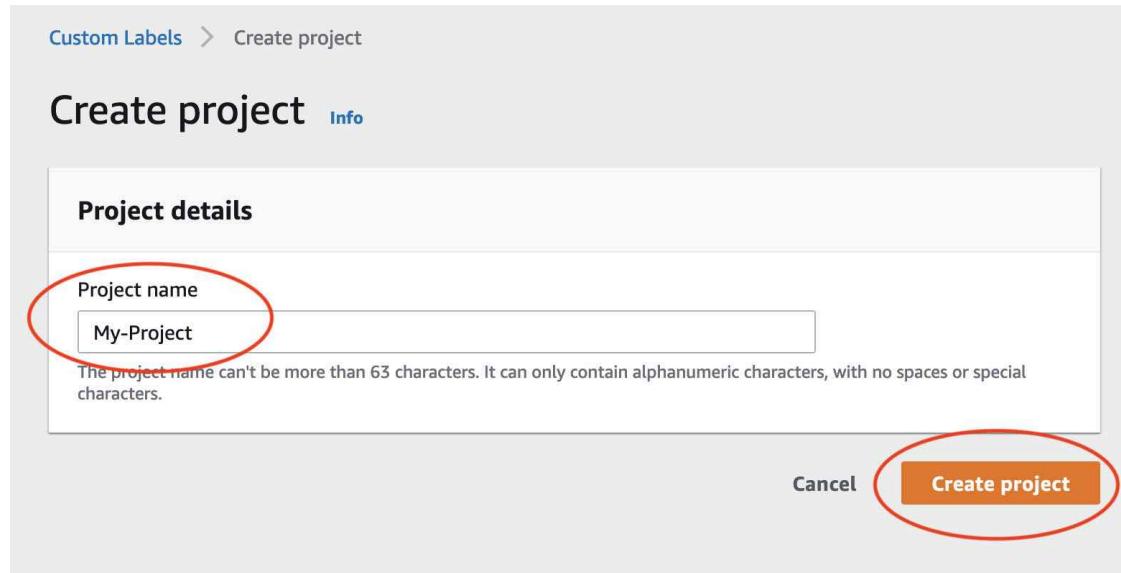
Step 3: Create a project

To manage your datasets and models you create a project. Each project should address a single use case, such as recognizing rooms in a house.

To create a project (console)

1. If you haven't already, set up the Amazon Rekognition Custom Labels console. For more information, see [Setting up Amazon Rekognition Custom Labels \(p. 4\)](#).
2. Sign in to the AWS Management Console and open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
3. In the left pane, choose **Use Custom Labels**. The Amazon Rekognition Custom Labels landing page is shown.
4. The Amazon Rekognition Custom Labels landing page, choose **Get started**

5. In the left navigation pane, choose **Projects**.
6. On the projects page, choose **Create Project**.
7. In **Project name**, enter a name for your project.
8. Choose **Create project** to create your project.



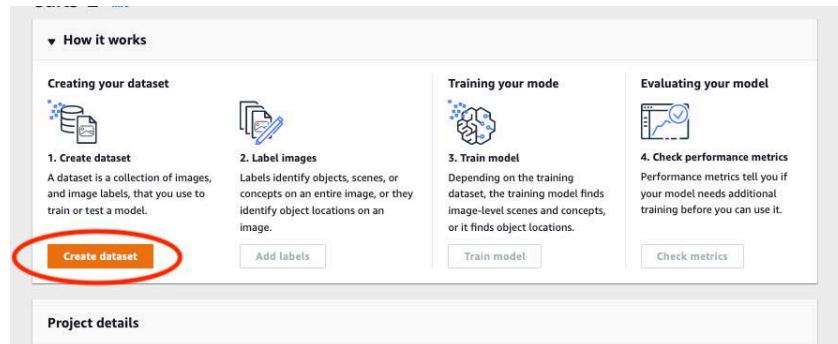
Step 4: Create training and test datasets

In this step you create a training dataset and a test dataset by uploading images from your local computer. You can upload as many as 30 images at a time. If you have a lot of images to upload, consider creating the datasets by importing the images from an Amazon S3 bucket. For more information, see [Amazon S3 bucket \(p. 59\)](#).

For more information about datasets, see [Managing datasets \(p. 211\)](#).

To create a dataset using images on a local computer (console)

1. On the project details page, choose **Create dataset**.



2. In the **Starting configuration** section, choose **Start with a training dataset and a test dataset**.
3. In the **Training dataset details** section, choose **Upload images from your computer**.
4. In the **Test dataset details** section, choose **Upload images from your computer**.
5. Choose **Create datasets**.

Create dataset [Info](#)

Starting configuration

Configuration options

Start with a single dataset
When you train your model, the dataset is split to create the training dataset (80%) and test dataset (20%) for your project.

Start with a training dataset and a test dataset
Recommended for most users. Start with the highest control over training, testing, and performance tuning.

What are training datasets and test datasets?

- A training dataset teaches your model to identify scenes or objects in images.
- A test dataset evaluates the performance of your trained model.

Training dataset details

Import images [Info](#)
Import images from one of the sources below.

Import Images from S3 bucket
Use images from an existing S3 bucket by entering the S3 bucket URL. You can automatically add labels based on your S3 bucket folder names.

Upload images from your computer
Add images by uploading files from your local computer. You're limited to uploading 30 images at one time.

Copy an existing Amazon Rekognition Custom Labels dataset
Use an existing dataset as a starting point for your new dataset. Your original dataset will remain unchanged.

Import images labeled by SageMaker Ground Truth
Provide the location of your manifest file. If you have a labeled datasets in a different format, convert them to a manifest format.

Test dataset details

Import images [Info](#)
Import images from one of the sources below.

Import Images from S3 bucket
Use images from an existing S3 bucket by entering the S3 bucket URL. You can automatically add labels based on your S3 bucket folder names.

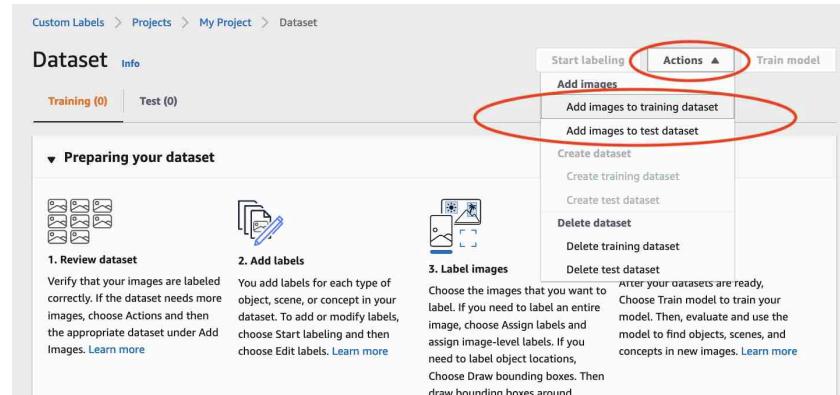
Upload images from your computer
Add images by uploading files from your local computer. You're limited to uploading 30 images at one time.

Copy an existing Amazon Rekognition Custom Labels dataset
Use an existing dataset as a starting point for your new dataset. Your original dataset will remain unchanged.

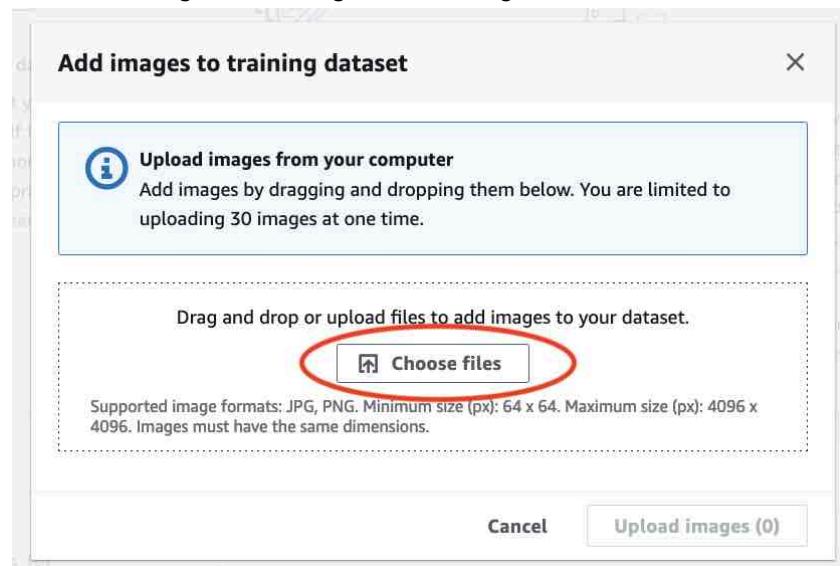
Import images labeled by SageMaker Ground Truth
Provide the location of your manifest file. If you have a labeled datasets in a different format, convert them to a manifest format.

[Cancel](#) [Create Datasets](#)

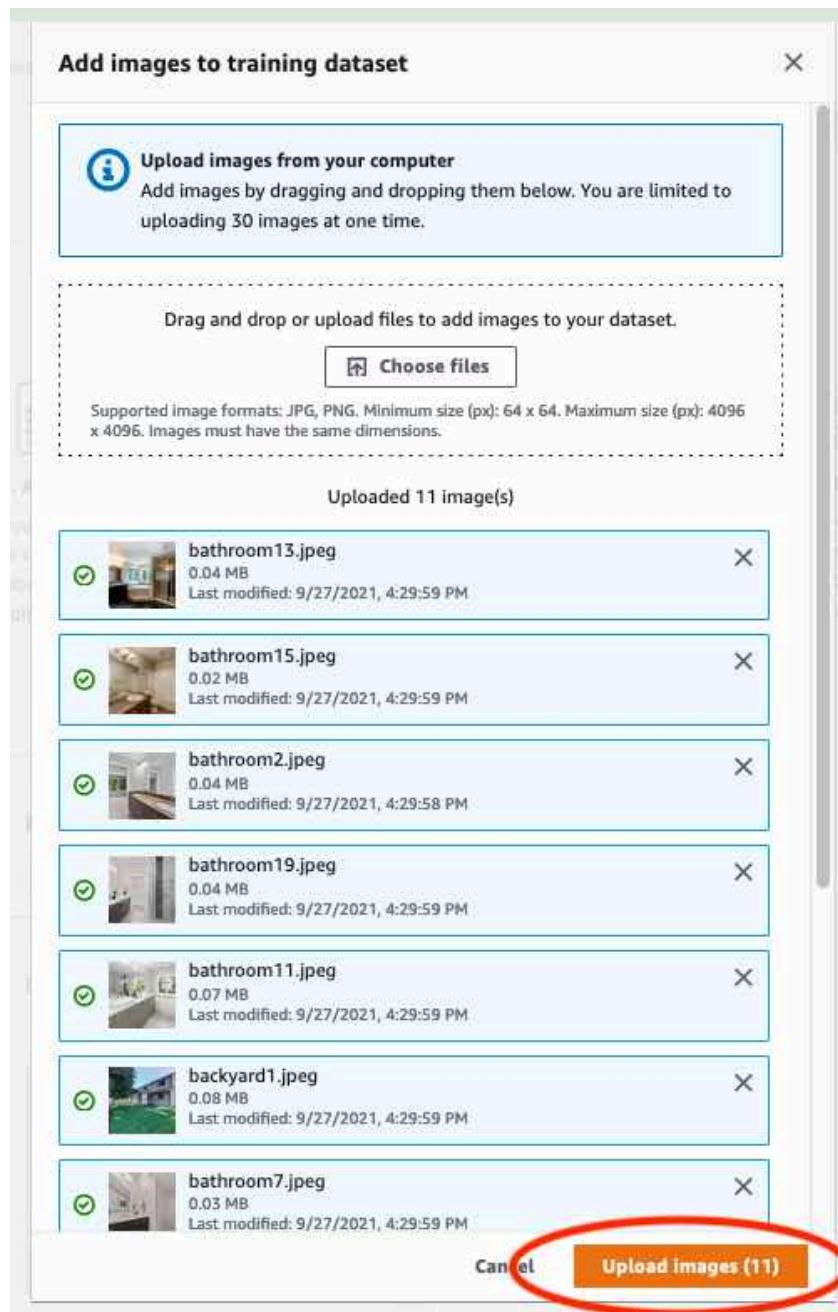
6. A dataset page appears with a **Training** tab and a **Test** tab for the respective datasets.
7. On the dataset page, choose the **Training** tab.
8. Choose **Actions** and then choose **Add images to training dataset**.



9. In the **Add images to training dataset** dialog box, choose **Choose files**.



10. Choose the images you want to upload to the dataset. You can upload as many as 30 images at a time.
11. Choose **Upload images**. It might take a few seconds for Amazon Rekognition Custom Labels to add the images to the dataset.



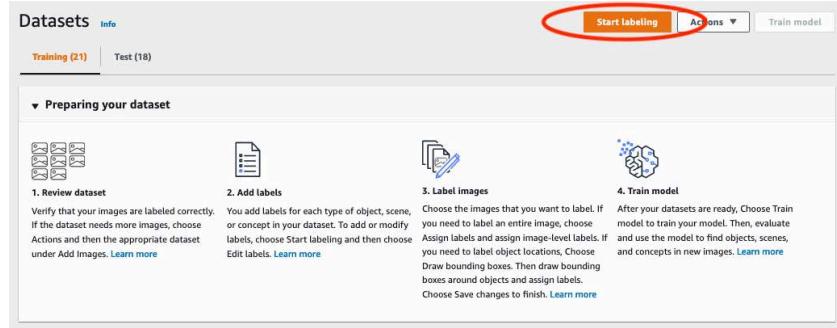
12. If you have more images to add to the training dataset, repeat steps 9-12.
13. Choose the **Test** tab.
14. Repeat steps 8 - 12 to add images to the test dataset. For step 8, choose **Actions** and then choose **Add images to test dataset**.

Step 5: Add labels to the project

In this step you add a label to the project for each of the classes you identified in step [Step 2: Decide your classes \(p. 35\)](#).

To add a new label (console)

1. On the dataset gallery page, choose **Start labeling** to enter labeling mode.



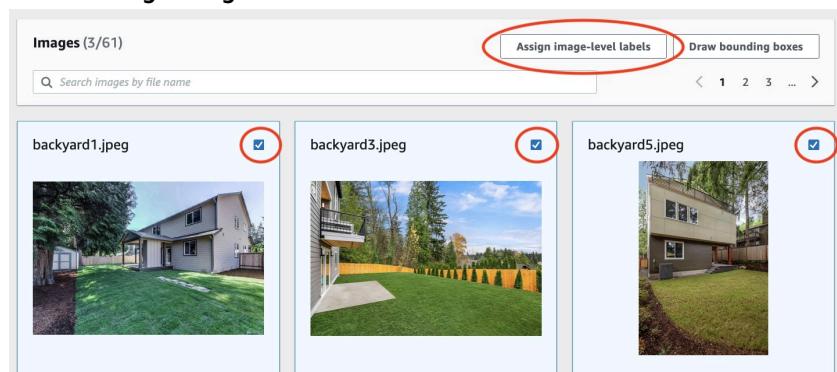
2. In the **Labels** section of the dataset gallery, choose **Edit labels** to open the **Manage labels** dialog box.
3. In the edit box, enter a new label name.
4. Choose **Add label**.
5. Repeat steps 3 and 4 until you have created all the labels you need.
6. Choose **Save** to save the labels that you added.

Step 6: Assign image-level labels to training and test datasets

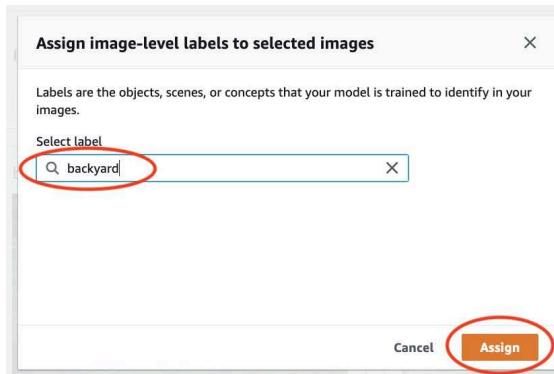
In this step you assign a single image level to each image in your training and test datasets. The image-level label is the class that each image represents.

To assign image-level labels to an image (console)

1. On the **Datasets** page, choose the **Training** tab.
2. Choose **Start labeling** to enter labeling mode.
3. Select one or more images that you want to add labels to. You can only select images on a single page at a time. To select a contiguous range of images on a page:
 - a. Select the first image.
 - b. Press and hold the shift key.
 - c. Select the second image. The images between the first and second image are also selected.
 - d. Release the shift key.
4. Choose **Assign image-level labels**.



5. In **Assign image-level labels to selected images** dialog box, select a label that you want to assign to the image or images.
6. Choose **Assign** to assign label to the image.



7. Repeat labeling until every image is annotated with the required labels.
8. Choose the **Test** tab.
9. Repeat steps to assign image level labels to the test dataset images.

Step 7: Train your model

Use the following steps to train your model. For more information, see [Training an Amazon Rekognition Custom Labels model \(p. 106\)](#).

To train your model (console)

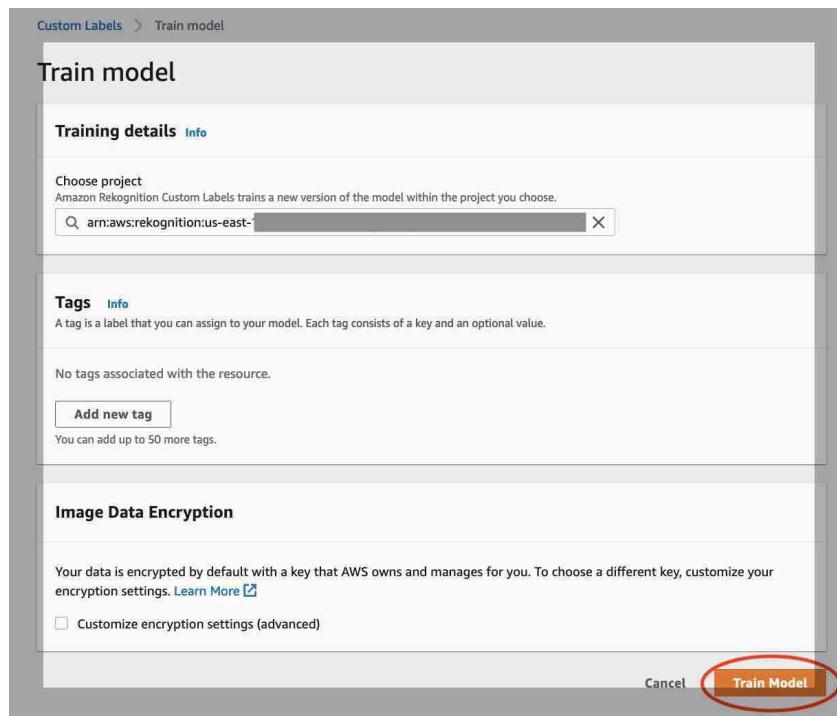
1. On the **Dataset** page, choose **Train model**.



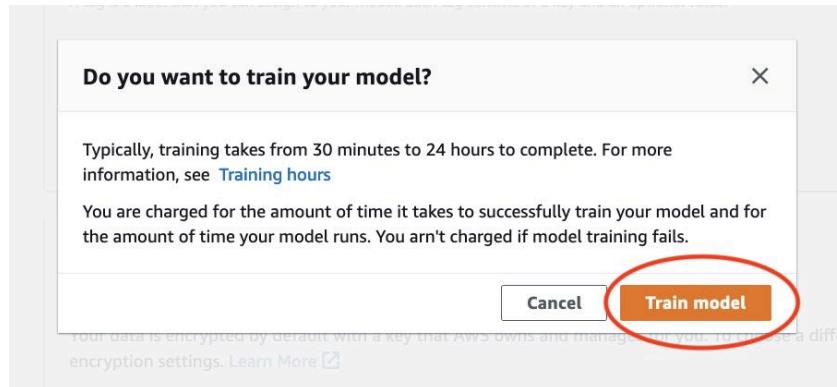
2. On the **Train model** page, choose **Train model**. The Amazon Resource Name (ARN) for your project is in the **Choose project** edit box.

Rekognition Custom Labels Guide

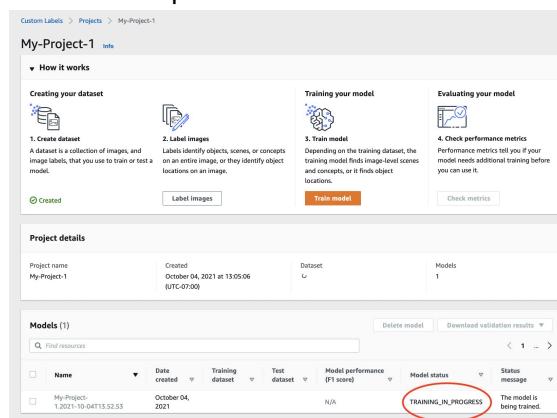
Step 7: Train your model



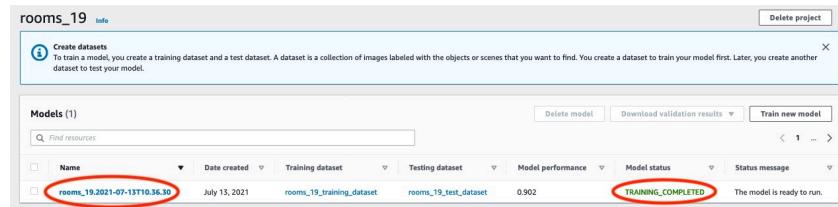
3. In the **Do you want to train your model?** dialog box, choose **Train model**.



4. In the **Models** section of the project page, you can see that training is in progress. You can check the current status by viewing the **Model Status** column for the model version. Training a model takes a while to complete.



5. After training completes, choose the model name. Training is finished when the model status is **TRAINING_COMPLETED**.

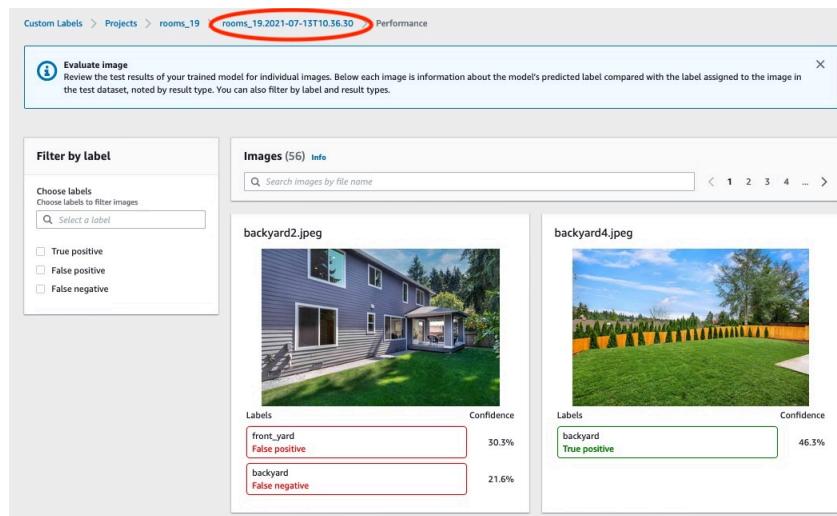


6. Choose the **Evaluate** button to see the evaluation results. For information about evaluating a model, see [Improving a trained Amazon Rekognition Custom Labels model \(p. 152\)](#).
7. Choose **View test results** to see the results for individual test images. For more information, see [Metrics for evaluating your model \(p. 152\)](#).

Evaluation results	
F1 score Info 0.902	Average precision Info 0.893
Date completed July 13, 2021 Trained in 1.223 hours	Overall recall Info 0.928
Training dataset 10 labels, 61 images	Testing dataset 10 labels, 56 images

Per label performance (10)					
Find labels					
Label name	F1 score	Test images	Precision	Recall	Assumed threshold
backyard	0.857	4	1.000	0.750	0.286
bathroom	0.889	9	0.889	0.889	0.185
bedroom	0.900	11	1.000	0.818	0.262
closet	1.000	2	1.000	1.000	0.169
entry_way	1.000	3	1.000	1.000	0.149
floor_plan	1.000	2	1.000	1.000	0.685

8. After viewing the test results, choose the project name to return to the model page.



Step 8: Start your model

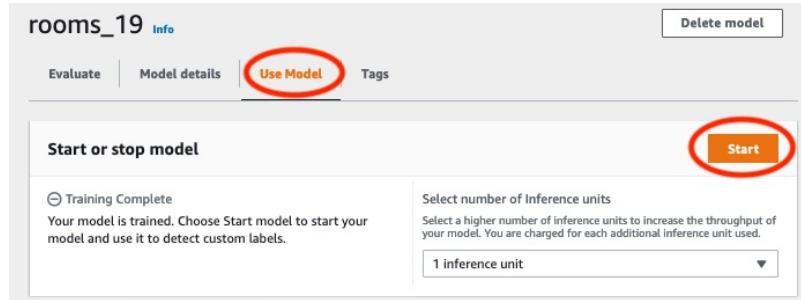
In this step you start your model. After your model starts, you can use it to analyze images.

You are charged for the amount of time that your model runs. Stop your model if you don't need to analyze images. You can restart your model at a later time. For more information, see [Running a trained Amazon Rekognition Custom Labels model \(p. 165\)](#).

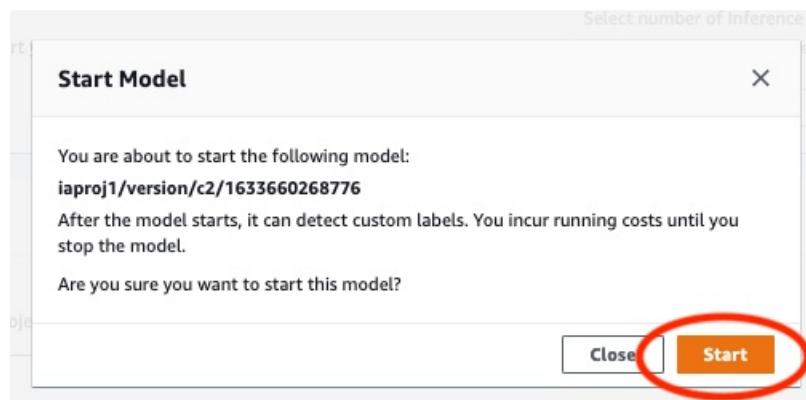
To start your model

1. Choose the **Use model** tab on the model page.
2. In the **Start or stop model** section do the following:

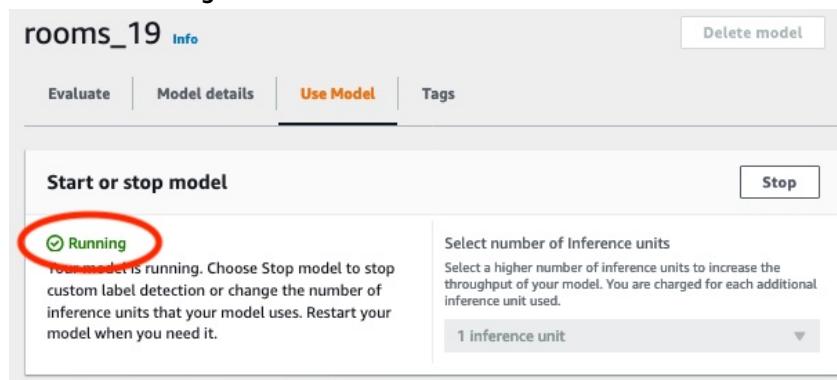
- a. Choose **Start**.



- b. In the **Start model** dialog box, choose **Start**.



3. Wait until the model is running. The model is running when the status in the **Start or stop model** section is **Running**.



Step 9: Analyze an image with your model

You analyze an image by calling the `DetectCustomLabels` API. In this step, you use the `detect-custom-labels` AWS Command Line Interface (AWS CLI) command to analyze an example image. You get the AWS CLI command from the Amazon Rekognition Custom Labels console. The console configures the AWS CLI command to use your model. You only need to supply an image that's stored in an Amazon S3 bucket.

Note

The console also provides Python example code.

The output from `detect-custom-labels` includes a list of labels found in the image, bounding boxes (if the model finds object locations), and the confidence that the model has in the accuracy of the predictions.

For more information, see [Analyzing an image with a trained model \(p. 182\)](#).

To analyze an image (console)

1. If you haven't already, set up the AWS CLI. For instructions, see [the section called "Step 3: Set Up the AWS CLI and AWS SDKs" \(p. 5\)](#).
2. Choose the **Use Model** tab and then choose **API code**.

The screenshot shows the 'Use Model' tab selected in the navigation bar of the AWS Rekognition Custom Labels console. The 'Start or stop model' section indicates the model is 'Running'. The 'Select number of Inference units' dropdown is set to '1 inference unit'. The 'Use your model' section includes an 'Amazon Resource Name (ARN)' input field and a '▶ API Code' button, which is circled in red.

3. Choose **AWS CLI command**.
4. In the **Analyze image** section, copy the AWS CLI command that calls `detect-custom-labels`.

The screenshot shows the 'Use your model' interface. At the top, there is a field for 'Amazon Resource Name (ARN)'. Below it, a section titled '▼ API Code' contains two options: 'AWS CLI command' (selected and circled in red) and 'Python'. Under 'AWS CLI command', there is a 'Start model' section with a command to start a project version. Under 'Analyze image', there is a command to detect custom labels in an image. Both sections include placeholder text for ARN and region.

```
aws rekognition start-project-version \
--project-version-arn "arn:aws:rekognition:us-east-1:MY_BUCKET" \
--min-inference-units 1 \
--region us-east-1

aws rekognition detect-custom-labels \
--project-version-arn "arn:aws:rekognition:us-east-1:MY_BUCKET" \
--image '{"S3Object": {"Bucket": "MY_BUCKET", "Name": "PATH_TO_MY_IMAGE"}' \
--region us-east-1
```

5. Upload an image to an Amazon S3 bucket. For instructions, see [Uploading Objects into Amazon S3](#) in the *Amazon Simple Storage Service User Guide*. If you're using images from the Rooms project, use one of the images you moved to a separate folder in [Step 1: Collect your images \(p. 34\)](#).
6. At the command prompt, enter the AWS CLI command that you copied in the previous step. It should look like the following example.

The value of `--project-version-arn` should be Amazon Resource Name (ARN) of your model. The value of `--region` should be the AWS Region in which you created the model.

Change `MY_BUCKET` and `PATH_TO_MY_IMAGE` to the Amazon S3 bucket and image that you used in the previous step.

```
aws rekognition detect-custom-labels \
--project-version-arn "model_arn" \
--image '{"S3Object": {"Bucket": "MY_BUCKET", "Name": "PATH_TO_MY_IMAGE"}' \
--region us-east-1
```

The JSON output from the AWS CLI command should look similar to the following. `Name` is the name of the image-level label that the model found. `Confidence` (0-100) is the model's confidence in the accuracy of the prediction.

```
{  
  "CustomLabels": [  
    {  
      "Name": "living_space",  
      "Confidence": 83.41299819946289  
    }  
  ]  
}
```

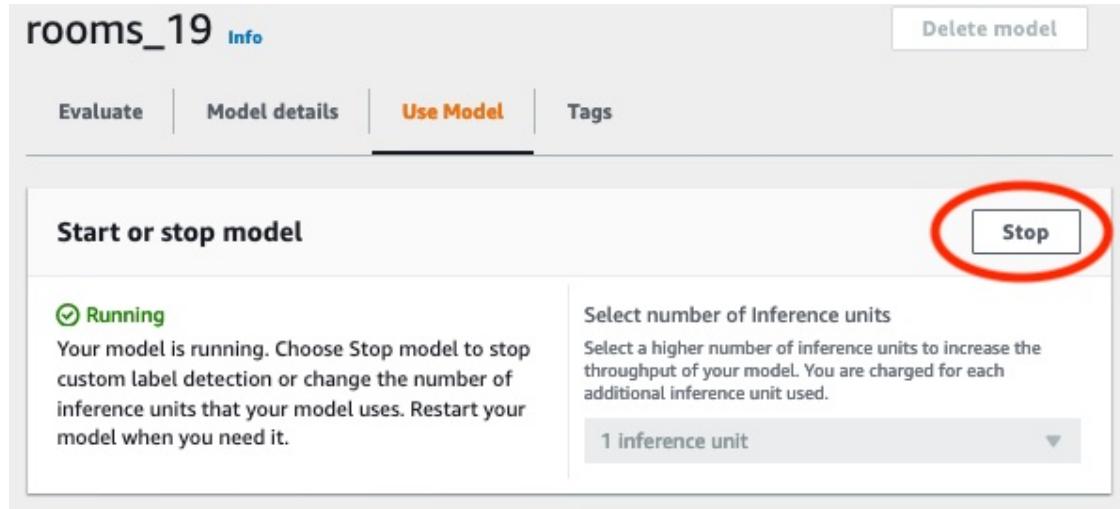
7. Continue to use the model to analyze other images. Stop the model if you are no longer using it.

Step 10: Stop your model

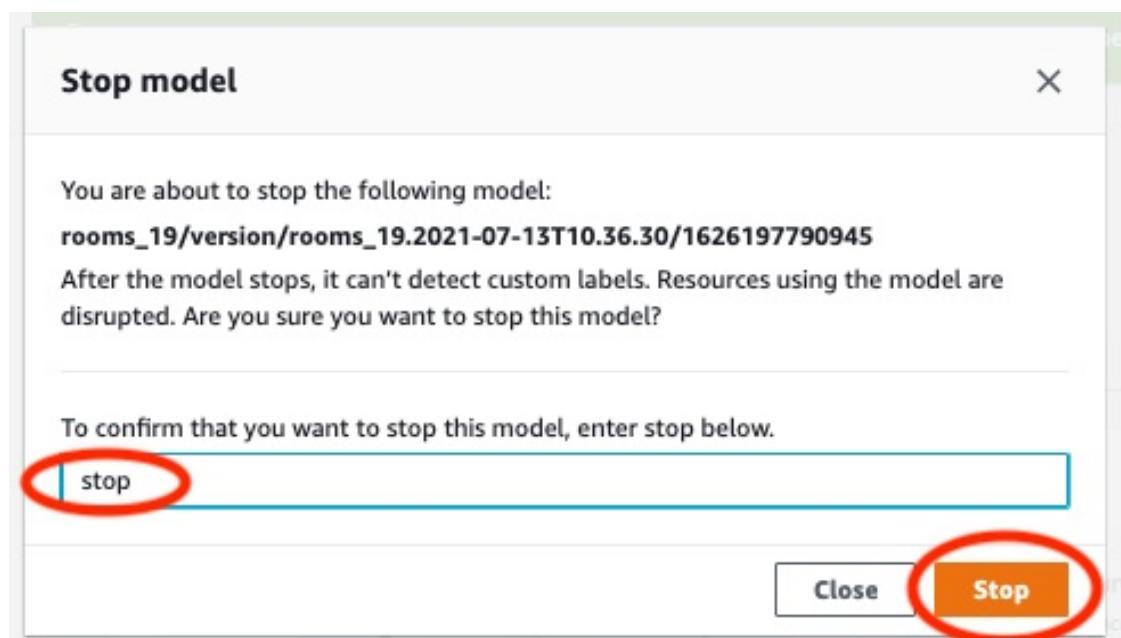
In this step you stop running your model. You are charged for the amount of time your model is running. If you have finished using the model, you should stop it.

To stop your model

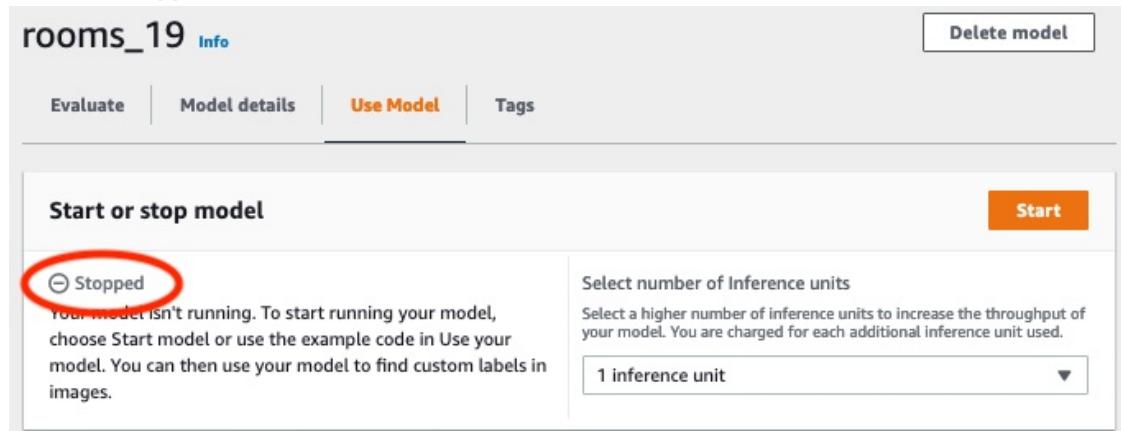
1. In the **Start or stop model** section choose **Stop**.



2. In the **Stop model** dialog box, enter **stop** to confirm that you want to stop the model.



3. Choose **Stop** to stop your model. The model has stopped when the status in the **Start or stop model** section is **Stopped**.



Creating an Amazon Rekognition Custom Labels model

A model is the software that you train to find the concepts, scenes, and objects that are unique to your business. You can create a model with the Amazon Rekognition Custom Labels console or with the AWS SDK. Before creating an Amazon Rekognition Custom Labels model, we recommend that you read [Understanding Amazon Rekognition Custom Labels \(p. 27\)](#).

This section provides console and SDK information about creating a project, creating training and test datasets for different model types, and training a model. Later sections show you how to improve and use your model. For a tutorial that shows you how to create and use a specific type of model with the console, see [Tutorial: Classifying images \(p. 34\)](#).

Topics

- [Creating a project \(p. 50\)](#)
- [Creating training and test datasets \(p. 54\)](#)
- [Training an Amazon Rekognition Custom Labels model \(p. 106\)](#)
- [Debugging a failed model training \(p. 116\)](#)

Creating a project

A project manages the model versions, training dataset, and test dataset for a model. You can create a project with the Amazon Rekognition Custom Labels console or with the API.

If you no longer need a project, you can delete it with the Amazon Rekognition Custom Labels console or with the API. For more information, see [Deleting an Amazon Rekognition Custom Labels model \(p. 268\)](#).

Topics

- [Creating an Amazon Rekognition Custom Labels Project \(Console\) \(p. 50\)](#)
- [Creating an Amazon Rekognition Custom Labels project \(SDK\) \(p. 51\)](#)

Creating an Amazon Rekognition Custom Labels Project (Console)

You can use the Amazon Rekognition Custom Labels console to create a project. The first time you use the console in a new AWS Region, Amazon Rekognition Custom Labels asks to create an Amazon S3 bucket (console bucket) in your account. The bucket is used to store project files. You can't use the Amazon Rekognition Custom Labels console unless the console bucket is created.

You can use the Amazon Rekognition Custom Labels console to create a project.

To create a project (console)

1. Sign in to the AWS Management Console and open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. In the left pane, choose **Use Custom Labels**. The Amazon Rekognition Custom Labels landing page is shown.
3. The Amazon Rekognition Custom Labels landing page, choose **Get started**.

4. In the left pane, Choose **Projects**.
5. Choose **Create Project**.
6. In **Project name**, enter a name for your project.
7. Choose **Create project** to create your project.
8. Follow the steps in [Creating training and test datasets \(p. 54\)](#) to create the training and test datasets for your project.

Creating an Amazon Rekognition Custom Labels project (SDK)

You create an Amazon Rekognition Custom Labels project by calling [CreateProject](#). The response is an Amazon Resource Name (ARN) that identifies the project. After you create a project, you create datasets for training and testing a model. For more information, see [Creating training and test datasets \(Console\) \(p. 59\)](#).

To create a project (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with AmazonRekognitionFullAccess permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following code to create a project.

AWS CLI

The following example creates a project and displays its ARN.

Change the value of project-name to the name of the project that you want to create.

```
aws rekognition create-project --project-name my_project
```

Python

The following example creates a project and displays its ARN. Supply the following command line arguments:

- project_name – the name of the project you want to create.

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsd़ocs/amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import argparse
import logging
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def create_project(rek_client, project_name):
    """
    Creates an Amazon Rekognition Custom Labels project
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    """

    # Create a new project
    response = rek_client.create_project(
        name=project_name
    )

    # Print the ARN of the new project
    print(f"Created project {project_name} with ARN {response['ProjectArn']}")
```

```
:param project_name: A name for the new project.
"""

try:
    #Create the project
    logger.info(f"Creating project: {project_name}")

    response=rek_client.create_project(ProjectName=project_name)

    logger.info(f"project ARN: {response['ProjectArn']}")

    return response['ProjectArn']

except ClientError as err:
    logger.exception(f"Couldn't create project - {project_name}: {err.response['Error']['Message']}")

raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_name", help="A name for the new project."
    )

def main():
    logging.basicConfig(level=logging.INFO, format"%(levelname)s: %(message)s")

    try:

        #get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Creating project: {args.project_name}")

        #Create the project
        rek_client=boto3.client('rekognition')

        project_arn=create_project(rek_client,
                                   args.project_name)

        print(f"Finished creating project: {args.project_name}")
        print(f"ARN: {project_arn}")

    except ClientError as err:
        logger.exception(f"Problem creating project: {err}")
        print(f"Problem creating project: {err}")

    if __name__ == "__main__":
        main()
```

Java V2

The following example creates a project and displays its ARN.

Supply the following command line argument:

- `project_name` – the name of the project you want to create.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.CreateProjectRequest;
import software.amazon.awssdk.services.rekognition.model.CreateProjectResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;

public class CreateProject {

    public static final Logger logger =
    Logger.getLogger(CreateProject.class.getName());

    public static String createMyProject(RekognitionClient rekClient, String
    projectName) {

        try {

            logger.log(Level.INFO, "Creating project: {0}", projectName);
            CreateProjectRequest createProjectRequest =
            CreateProjectRequest.builder().projectName(projectName).build();

            CreateProjectResponse response =
            rekClient.createProject(createProjectRequest);

            logger.log(Level.INFO, "Project ARN: {0} ", response.projectArn());

            return response.projectArn();

        } catch (RekognitionException e) {
            logger.log(Level.SEVERE, "Could not create project: {0}",
            e.getMessage());
            throw e;
        }
    }

    public static void main(String args[]) {
        final String USAGE = "\n" + "Usage: " + "<project_name> <bucket> <image>\n"
        "\n" + "Where:\n" +
        "    + "    project_name - A name for the new project\n\n";
        if (args.length != 1) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String projectName = args[0];
        String projectArn = null;
        ;

        try {
```

```
// Get the Rekognition client
RekognitionClient rekClient = RekognitionClient.builder().build();

// Create the project
projectArn = createMyProject(rekClient, projectName);

System.out.println(String.format("Created project: %s %nProject ARN:
%s", projectName, projectArn));

rekClient.close();

} catch (RekognitionException rekError) {
    logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
    System.exit(1);
}

}
```

3. Note the name of the project ARN that's displayed in the response. You'll need it to create a model.
4. Follow the steps in [Create training and test datasets \(SDK\) \(p. 65\)](#) to create the training and test datasets for your project.

Creating training and test datasets

A dataset is a set of images and labels that describe those images. Your project needs a training dataset and a test dataset. Amazon Rekognition Custom Labels uses the training dataset to train your model. After training, Amazon Rekognition Custom Labels uses the test dataset to verify how well the trained model predicts the correct labels.

You can create datasets with the Amazon Rekognition Custom Labels console or with the AWS SDK. Before creating a dataset, we recommend reading [Understanding Amazon Rekognition Custom Labels \(p. 27\)](#).

The steps creating training and tests datasets for a project are:

To create training and test datasets for your project

1. Determine how you need to label your training and test datasets. For more information, [Purposing datasets \(p. 55\)](#).
2. Collect the images for your training and test datasets. For more information, see [the section called "Preparing images" \(p. 58\)](#).
3. Create the training and test datasets. For more information, see [Creating training and test datasets \(Console\) \(p. 59\)](#). If you're using the AWS SDK, see [Create training and test datasets \(SDK\) \(p. 65\)](#).
4. If necessary, add image-level labels or bounding boxes to your dataset images. For more information, see [Labeling images \(p. 99\)](#).

Topics

- [Purposing datasets \(p. 55\)](#)
- [Preparing images \(p. 58\)](#)
- [Creating training and test datasets \(Console\) \(p. 59\)](#)

- [Create training and test datasets \(SDK\) \(p. 65\)](#)
- [Debugging datasets \(p. 95\)](#)
- [Labeling images \(p. 99\)](#)

Purposing datasets

How you label the training and test datasets in your project determines the type of model that you create. With Amazon Rekognition Custom Labels you can create models that do the following.

- [Find objects, scenes, and concepts \(p. 55\)](#)
- [Find object locations \(p. 56\)](#)
- [Find brand locations \(p. 56\)](#)

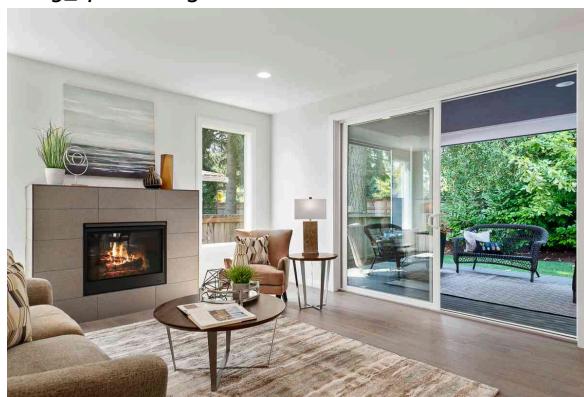
Find objects, scenes, and concepts

The model classifies the objects, scenes, and concepts that are associated with an entire image.

You can create two types of classification model, *image classification* and *multi-label classification*. For both types of classification model, the model finds one or more matching labels from the complete set of labels used for training. The training and test datasets both require at least two labels.

Image classification

The model classifies images as belonging to a set of predefined labels. For example, you might want a model that determines if an image contains a living space. The following image might have a *living_space* image-level label.



For this type of model, add a single image-level label to each of the training and test dataset images. For an example project, see [Image classification \(p. 12\)](#).

Multi-label classification

The model classifies images into multiple categories, such as the type of flower and whether it has leaves, or not. For example, the following image might have *mediterranean_spurge* and *no_leaves* image level labels.



For this type of model assign image-level labels for each category to the training and test dataset images. For an example project, see [Multi-label image classification \(p. 12\)](#).

Assigning image-level labels

If your images are stored in an Amazon S3 bucket, you can use [folder names \(p. 59\)](#) to automatically add image-level labels. For more information, see [Amazon S3 bucket \(p. 59\)](#). You can also add image-level labels to images after you create a dataset. For more information, see [the section called "Assigning image-level labels to an image" \(p. 102\)](#). You can add new labels as you need them. For more information, see [Managing labels \(p. 100\)](#).

Find object locations

To create a model that predicts the location of objects in your images, you define object location bounding boxes and labels for the images in your training and test datasets. A bounding box is a box that tightly surrounds an object. For example, the following image shows bounding boxes around an Amazon Echo and an Amazon Echo Dot. Each bounding box has an assigned label (*Amazon Echo* or *Amazon Echo Dot*).



To find object locations, your datasets needs at least one label. During model training, a further label is automatically created that represents the area outside of the bounding boxes on an image.

Assigning bounding boxes

When you create your dataset, you can include bounding box information for your images. For example, you can import a SageMaker Ground Truth format [manifest file \(p. 242\)](#) that contains bounding boxes. You can also add bounding boxes after you create a dataset. For more information, see [Labeling objects with bounding boxes \(p. 104\)](#). You can add new labels as you need them. For more information, see [Managing labels \(p. 100\)](#).

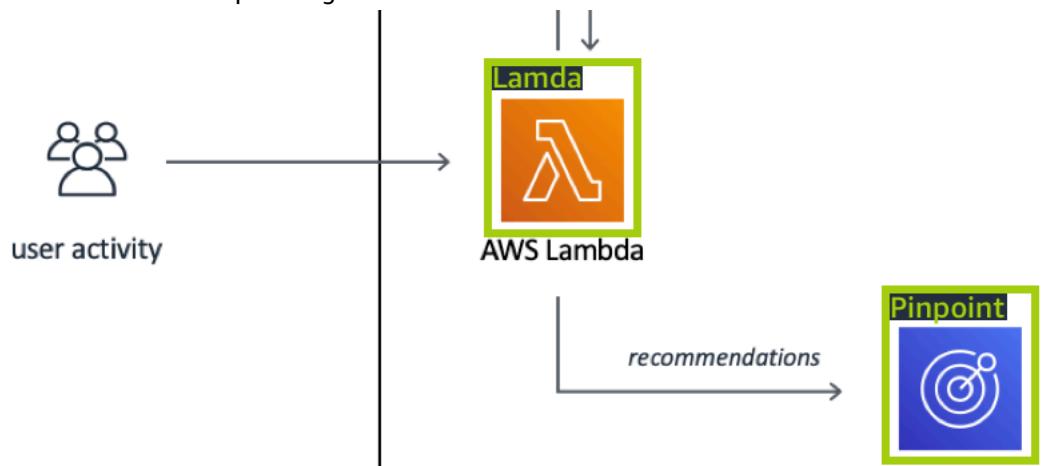
Find brand locations

If you want to find the location of brands, such as logos and animated characters, you can use two different types of images for your training dataset images.

- Images that are of the logo only. Each image needs a single image-level label that represents the logo name. For example, the image-level label for the following image could be *Lambda*.



- Images that contain the logo in natural locations, such as a football game or an architectural diagram. Each training image needs bounding boxes that surround each instance of the logo. For example, the following image shows an architectural diagram with labeled bounding boxes surrounding the AWS Lambda and Amazon Pinpoint logos.



We recommend that you don't mix image-level labels and bounding boxes in your training images.

The test images must have bounding boxes around instances of the brand that you want to find. You can split the training dataset to create the test dataset, only if the training images include labeled bounding boxes. If the training images only have image-level labels, you must create a test dataset set that includes images with labeled bounding boxes. If you train a model to find brand locations, do [Labeling objects with bounding boxes \(p. 104\)](#) and [Assigning image-level labels to an image \(p. 102\)](#) according to how you label your images.

The [Brand detection \(p. 12\)](#) example project shows how Amazon Rekognition Custom Labels uses labeled bounding boxes to train a model that finds object locations.

Label requirements for model types

Use the following table to determine how to label your images.

You can combine image-level labels and bounding box labeled images in a single dataset. In this case, Amazon Rekognition Custom Labels chooses whether to create an image-level model or an object location model.

Example	Training images	Test images
Image classification (p. 55)	1 Image-level label per image	1 Image-level label per image
Multi-label classification (p. 55)	Multiple image-level labels per image	Multiple image-level labels per image
Find brand locations (p. 56)	image level-labels (you can also use Labeled bounding boxes)	Labeled bounding boxes
Find object locations (p. 56)	Labeled bounding boxes	Labeled bounding boxes

Preparing images

The images in your training and test dataset contain the objects, scenes, or concepts that you want your model to find.

The content of images should be in a variety of backgrounds and lighting that represent the images that you want the trained model to identify.

This section provides information about the images in your training and test dataset.

Image format

You can train Amazon Rekognition Custom Labels models with images that are in PNG and in JPEG format. Similarly, to detect custom labels using `DetectCustomLabels`, you need images that are in PNG and JPEG format.

Input image recommendations

Amazon Rekognition Custom Labels requires images to train and test your model. To prepare your images, consider following:

- Choose a specific domain for the model you want to create. For example, you could choose a model for scenic views and another model for objects such as machine parts. Amazon Rekognition Custom Labels works best if your images are in the chosen domain.
- Use at least 10 images to train your model.
- Images must be in PNG or JPEG format.
- Use images that show the object in a variety of lightings, backgrounds, and resolutions.
- Training and testing images should be similar to the images that you want to use the model with.
- Decide what labels to assign to the images.
- Ensure that images are sufficiently large in terms of resolution. For more information, see [Guidelines and quotas in Amazon Rekognition Custom Labels \(p. 323\)](#).
- Ensure that occlusions don't obscure objects that you want to detect.
- Use images that have sufficient contrast with the background.
- Use images that are bright and sharp. Avoid using images that may be blurry due to subject and camera motion as much as possible.
- Use an image where the object occupies a large proportion of the image.

- Images in your test dataset shouldn't be images that are in the training dataset. They should include the objects, scenes, and concepts that the model is trained to analyze.

Image set size

Amazon Rekognition Custom Labels uses a set of images to train a model. At a minimum, you should use at least 10 images for training. Amazon Rekognition Custom Labels stores training and testing images in datasets. For more information, see [Creating training and test datasets \(Console\) \(p. 59\)](#).

Creating training and test datasets (Console)

You can start with a project that has a single dataset, or a project that has separate training and test datasets. If you start with a single dataset, Amazon Rekognition Custom Labels splits your dataset during training to create a training dataset (80%) and a test dataset (%20) for your project. Start with a single dataset if you want Amazon Rekognition Custom Labels to decide where images are used for training and testing. For complete control over training, testing, and performance tuning, we recommend that you start your project with separate training and test datasets.

You can create training and test datasets for a project by importing images from one of the following locations:

- [Amazon S3 bucket \(p. 59\)](#)
- [Local computer \(p. 61\)](#)
- [Amazon SageMaker Ground Truth \(p. 62\)](#)
- [Existing dataset \(p. 64\)](#)

If you start your project with separate training and test datasets, you can use different source locations for each dataset.

Depending on where you import your images from, your images might be unlabeled. For example, images imported from a local computer aren't labeled. Images imported from an Amazon SageMaker Ground Truth manifest file are labeled. You can use the Amazon Rekognition Custom Labels console to add, change, and assign labels. For more information, see [Labeling images \(p. 99\)](#).

If images are uploading with errors, images are missing, or labels are missing from images, read [Debugging a failed model training \(p. 116\)](#).

For more information about datasets, see [Managing datasets \(p. 211\)](#).

Amazon S3 bucket

The images are imported from an Amazon S3 bucket. You can use the console bucket, or another Amazon S3 bucket in your AWS account. If you are using the console bucket, the required permissions are already set up. If you are not using the console bucket, see [Accessing external Amazon S3 Buckets \(p. 7\)](#).

During dataset creation, you can choose to assign label names to images based on the name of the folder that contains the images. The folder(s) must be a child of the Amazon S3 folder path that you specify in **S3 folder location** during dataset creation. To create a dataset, see [Creating a dataset by importing images from an S3 bucket \(p. 60\)](#).

For example, assume the following folder structure in an Amazon S3 bucket. If you specify the Amazon S3 folder location as `S3-bucket/alexa-devices`, the images in the folder `echo` are assigned the label `echo`. Similarly, images in the folder `echo-dot` are assigned the label `echo-dot`. The names of deeper child folders aren't used to label images. Instead, the appropriate child folder of the Amazon S3 folder

location is used. For example, images in the folder *white-echo-dots* are assigned the label *echo-dot*. Images at the level of the S3 folder location (*alexa-devices*) don't have labels assigned to them.

Folders deeper in the folder structure can be used to label images by specifying a deeper S3 folder location. For example, If you specify *S3-bucket/alexa-devices/echo-dot*, Images in the folder *white-echo-dot* are labeled *white-echo-dot*. Images outside the specified s3 folder location, such as *echo*, aren't imported.

```
S3-bucket
### alexa-devices
### echo
#   ###
#   ###
#   .
#   .
### echo-dot
### white-echo-dot
#   ###
#   white-echo-dot-image-1.png
#   white-echo-dot-image-2.png
#
### echo-dot-image-1.png
### echo-dot-image-2.png
### .
### .
```

We recommend that you use the Amazon S3 bucket (console bucket) created for you by Amazon Rekognition when you first opened the console in the current AWS region. If the Amazon S3 bucket that you are using is different (external) to the console bucket, the console prompts you to set up appropriate permissions during dataset creation. For more information, see [the section called "Step 4: Set Up Permissions" \(p. 6\)](#).

Creating a dataset by importing images from an S3 bucket

The following procedure shows you how to create a dataset using images stored in the Console S3 bucket. The images are automatically labeled with the name of the folder in which they are stored.

After you have imported your images, you can add more images, assign labels, and add bounding boxes from a dataset's gallery page. For more information, see [Labeling images \(p. 99\)](#).

Upload your images to an Amazon Simple Storage Service bucket

1. Create a folder on your local file system. Use a folder name such as *alexa-devices*.
2. Within the folder you just created, create folders named after each label that you want to use. For example, *echo* and *echo-dot*. The folder structure should be similar to the following.

```
alexa-devices
### echo
#   ###
#   ###
#   .
#   .
### echo-dot
### echo-dot-image-1.png
### echo-dot-image-2.png
### .
### .
```

3. Place the images that correspond to a label into the folder with the same label name.
4. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.

5. **Add the folder** you created in step 1 to the Amazon S3 bucket (console bucket) created for you by Amazon Rekognition Custom Labels during *First Time Set Up*. For more information, see [Managing an Amazon Rekognition Custom Labels project \(p. 200\)](#).
6. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
7. Choose **Use Custom Labels**.
8. Choose **Get started**.
9. In the left navigation pane, choose **Projects**.
10. In the **Projects** page, choose the project to which you want to add a dataset. The details page for your project is displayed.
11. Choose **Create dataset**. The **Create dataset** page is shown.
12. In **Starting configuration**, choose either **Start with a single dataset** or **Start with a training dataset**. To create a higher quality model, we recommend starting with separate training and test datasets.

Single dataset

- a. In the **Training dataset details** section, choose **Import images from S3 bucket**.
- b. In the **Training dataset details** section, Enter the information for steps 13 - 15 in the **Image source configuration** section.

Separate training and test datasets

- a. In the **Training dataset details** section, choose **Import images from S3 bucket**.
- b. In the **Training dataset details** section, enter the information for steps 13 - 15 in the **Image source configuration** section.
- c. In the **Test dataset details** section, choose **Import images from S3 bucket**.
- d. In the **Test dataset details** section, enter the information for steps 13 - 15 in the **Image source configuration** section.
13. Choose **Import images from Amazon S3 bucket**.
14. In **S3 URI**, enter the Amazon S3 bucket location and folder path.
15. Choose **Automatically attach labels to images based on the folder**.
16. Choose **Create Datasets**. The datasets page for your project opens.
17. If you need to add or change labels, do [Labeling images \(p. 99\)](#).
18. Follow the steps in [Training a model \(Console\) \(p. 107\)](#) to train your model.

Local computer

The images are loaded directly from your computer. You can upload up to 30 images at a time.

The images you upload won't have labels associated with them. For more information, see [Labeling images \(p. 99\)](#).

Note

If you have many images to upload, consider using an Amazon S3 bucket. For more information, see [Amazon S3 bucket \(p. 59\)](#).

To create a dataset using images on a local computer (console)

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. Choose **Use Custom Labels**.
3. Choose **Get started**.
4. In the left navigation pane, choose **Projects**.

5. In the **Projects** page, choose the project to which you want to add a dataset. The details page for your project is displayed.
6. Choose **Create dataset**. The **Create dataset** page is shown.
7. In **Starting configuration**, choose either **Start with a single dataset** or **Start with a training dataset**. To create a higher quality model, we recommend starting with separate training and test datasets.

Single dataset

- a. In the **Training dataset details section** section, choose **Upload images from your computer**.
- b. Choose **Create Dataset**.
- c. On the project's dataset page, choose **Add images**.
- d. Choose the images you want to upload into the dataset from your computer files. You can drag the images or choose the images that you want to upload from your local computer.
- e. Choose **Upload images**.

Separate training and test datasets

- a. In the **Training dataset details section**, choose **Upload images from your computer**.
- b. In the **Test dataset details section**, choose **Upload images from your computer**.

Note

Your training and test datasets can have different image sources.

- c. Choose **Create Datasets**. Your project's datasets page appears with a **Training** tab and a **Test** tab for the respective datasets.
- d. Choose **Actions** and then choose **Add images to training dataset**.
- e. Choose the images you want to upload to the dataset. You can drag the images or choose the images that you want to upload from your local computer.
- f. Choose **Upload images**.
- g. Repeat steps 5e - 5g. For step 5e, choose **Actions** and then choose **Add images to test dataset**.

8. Follow the steps in [Labeling images \(p. 99\)](#) to label your images.
9. Follow the steps in [Training a model \(Console\) \(p. 107\)](#) to train your model.

Amazon SageMaker Ground Truth

You can create a dataset using an Amazon SageMaker Ground Truth format manifest file. You can use the manifest file from an Amazon SageMaker Ground Truth job, or you can create your own manifest file. To create your own manifest file, see [Creating a manifest file \(p. 242\)](#).

With Amazon SageMaker 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 that allows you to create a labeled set of images. Amazon Rekognition Custom Labels imports SageMaker Ground Truth manifest files from an Amazon S3 bucket that you specify.

Amazon Rekognition Custom Labels supports the following SageMaker Ground Truth tasks.

- [Image Classification](#)
- [Bounding Box](#)

The files you import are the images and a manifest file. The manifest file contains label and bounding box information for the images you import.

If your images and labels aren't in the format of a SageMaker Ground Truth manifest file, you can create a SageMaker format manifest file and use it to import your labeled images. For more information, see [Creating a manifest file \(p. 242\)](#).

Amazon Rekognition needs permissions to access the Amazon S3 bucket where your images are stored. If you are using the console bucket set up for you by Amazon Rekognition Custom Labels, the required permissions are already set up. If you are not using the console bucket, see [Accessing external Amazon S3 Buckets \(p. 7\)](#).

Topics

- [Creating a dataset with a SageMaker Groundtruth job \(Console\) \(p. 63\)](#)
- [External \(p. 64\)](#)

Creating a dataset with a SageMaker Groundtruth job (Console)

The following procedure shows you how to create a dataset by using images labeled by a SageMaker Ground Truth job. The job output files are stored in your Amazon Rekognition Custom Labels console bucket.

To create a dataset using images labeled by a SageMaker Ground Truth job (console)

1. Sign in to the AWS Management Console and open the Amazon S3 console at <https://console.aws.amazon.com/s3/>.
2. In the console bucket, [create a folder](#) to hold your training images.

Note
The console bucket is created when you first open the Amazon Rekognition Custom Labels console in an AWS Region. For more information, see [Managing an Amazon Rekognition Custom Labels project \(p. 200\)](#).
3. [Upload your images](#) to the folder that you just created.
4. In the console bucket, create a folder to hold the output of the Ground Truth job.
5. Open the SageMaker console at <https://console.aws.amazon.com/sagemaker/>.
6. Create a Ground Truth labeling job. You'll need the Amazon S3 URLs for the folders you created in step 2 and step 4. For more information, see [Use Amazon SageMaker Ground Truth for Data Labeling](#).
7. Repeat steps 1 - 6 to create SageMaker Ground Truth job for your test dataset.
8. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
9. Choose **Use Custom Labels**.
10. Choose **Get started**.
11. In the left navigation pane, choose **Projects**.
12. In the **Projects** page, choose the project to which you want to add a dataset. The details page for your project is displayed.
13. Choose **Create dataset**. The **Create dataset** page is shown.
14. In **Starting configuration**, choose either **Start with a single dataset** or **Start with a training dataset**. To create a higher quality model, we recommend starting with separate training and test datasets.

Single dataset

- a. In the **Training dataset details** section, choose **Import images labeled by SageMaker Ground Truth**.
- b. In **.manifest file location** enter the location of the output.manifest file in the folder you created in step 4. It should be in the sub-folder **Ground-Truth-Job-Name/manifests/output**.

- c. Choose **Create Dataset**. The datasets page for your project opens.

Separate training and test datasets

- a. In the **Training dataset details** section, choose **Import images labeled by SageMaker Ground Truth**.
- b. In **.manifest file location** enter the location of the output.manifest file in the folder you created in step 4. It should be in the sub-folder *Ground-Truth-Job-Name*/manifests/output.
- c. In the **Test dataset details** section, choose **Import images labeled by SageMaker Ground Truth**.

Note

Your training and test datasets can have different image sources.

- d. In **.manifest file location** enter the location of the output.manifest file in the folder you created for the test images (step 7). It should be in the sub-folder *Ground-Truth-Job-Name*/manifests/output.
- e. Choose **Create Datasets**. The datasets page for your project opens.

15. If you need to add or change labels, do [Labeling images \(p. 99\)](#).
16. Follow the steps in [Training a model \(Console\) \(p. 107\)](#) to train your model.

External

Existing dataset

If you've previously created a dataset, you can copy its contents to a new dataset.

To create a dataset using an existing Amazon Rekognition Custom Labels dataset (console)

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. Choose **Use Custom Labels**.
3. Choose **Get started**.
4. In the left navigation pane, choose **Projects**.
5. In the **Projects** page, choose the project to which you want to add a dataset. The details page for your project is displayed.
6. Choose **Create dataset**. The **Create dataset** page is shown.
7. In **Starting configuration**, choose either **Start with a single dataset** or **Start with a training dataset**. To create a higher quality model, we recommend starting with separate training and test datasets.

Single dataset

- a. In the **Training dataset details** section, choose **Copy an existing Amazon Rekognition Custom Labels dataset**.
- b. In the **Training dataset details** section, in the **Dataset** edit box, type or select the name of the dataset that you want to copy.
- c. Choose **Create Dataset**. The datasets page for your project opens.

Separate training and test datasets

- a. In the **Training dataset details** section, choose **Copy an existing Amazon Rekognition Custom Labels dataset**.

- b. In the **Training dataset details** section, in the **Dataset** edit box, type or select the name of the dataset that you want to copy.
- c. In the **Test dataset details** section, choose **Copy an existing Amazon Rekognition Custom Labels dataset**.
- d. In the **Test dataset details** section, in the **Dataset** edit box, type or select the name of the dataset that you want to copy.

Note

Your training and test datasets can have different image sources.

- e. Choose **Create Datasets**. The datasets page for your project opens.
8. If you need to add or change labels, do [Labeling images \(p. 99\)](#).
9. Follow the steps in [Training a model \(Console\) \(p. 107\)](#) to train your model.

Create training and test datasets (SDK)

To train an Amazon Rekognition Custom Labels model, your project needs a training dataset and a test dataset.

Topics

- [Training dataset \(p. 65\)](#)
- [Test dataset \(p. 65\)](#)
- [Creating a dataset with a manifest file \(SDK\) \(p. 66\)](#)
- [Creating a dataset using an existing dataset \(SDK\) \(p. 72\)](#)
- [Creating an empty dataset \(SDK\) \(p. 77\)](#)
- [Adding more images \(SDK\) \(p. 83\)](#)
- [Splitting a training dataset to create a test dataset \(SDK\) \(p. 88\)](#)

Training dataset

You can use the AWS SDK to create a training dataset in the following ways.

- Use [CreateDataset](#) with an Amazon Sagemaker format manifest file that you provide. For more information, see [the section called "Creating a manifest file" \(p. 242\)](#). For example code, see [Creating a dataset with a manifest file \(SDK\) \(p. 66\)](#).
- Use [CreateDataset](#) to copy an existing Amazon Rekognition Custom Labels dataset. For example code, see [Creating a dataset using an existing dataset \(SDK\) \(p. 72\)](#).
- Create an empty dataset with [CreateDataset](#) and add dataset entries at a later time with [UpdateDatasetEntries](#). To create an empty dataset, see [Adding a dataset to a project \(p. 212\)](#). To add images to a dataset, see [Adding more images \(SDK\) \(p. 218\)](#). You need to add the dataset entries before you can train a model.

Test dataset

You can use the AWS SDK to create a test dataset in the following ways:

- Use [CreateDataset](#) with an Amazon Sagemaker format manifest file that you provide. For more information, see [the section called "Creating a manifest file" \(p. 242\)](#). For example code, see [Creating a dataset with a manifest file \(SDK\) \(p. 66\)](#).
- Use [CreateDataset](#) to copy an existing Amazon Rekognition Custom Labels dataset. For example code, see [Creating a dataset using an existing dataset \(SDK\) \(p. 72\)](#).

- Create an empty dataset with `CreateDataset` and add dataset entries at a later time with `UpdateDatasetEntries`. To create an empty dataset, see [Adding a dataset to a project \(p. 212\)](#). To add images to a dataset, see [Adding more images \(SDK\) \(p. 218\)](#). You need to add the dataset entries before you can train a model.
- Split the training dataset into separate training and test datasets. First create an empty test dataset with `CreateDataset`. Then move 20% of the training dataset entries into the test dataset by calling `DistributeDatasetEntries`. To create an empty dataset, see [Adding a dataset to a project \(SDK\) \(p. 212\)](#). To split the training dataset, see [Splitting a training dataset to create a test dataset \(SDK\) \(p. 88\)](#).

Creating a dataset with a manifest file (SDK)

The following procedure shows you how to create training or test datasets from a manifest using the `CreateDataset` API.

You can use an existing manifest file, such as the output from an SageMaker Groundtruth job, or create your own manifest file. To create your own manifest file, see [the section called “Creating a manifest file” \(p. 242\)](#).

1. If you haven't already:
 - a. Create or update an IAM user with `AmazonRekognitionFullAccess` and permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following example code to create the training and test dataset.

AWS CLI

Use the following code to create a dataset. Replace the following:

- `project_arn` — the ARN of the project that you want to add the test dataset to.
- `type` — the type of dataset that you want to create (TRAIN or TEST)
- `bucket` — the bucket that contains the manifest file for the dataset.
- `manifest_file` — the path and file name of the manifest file.

```
aws rekognition create-dataset --project-arn project_arn \  
  --dataset-type type \  
  --dataset-source '{ "GroundTruthManifest": { "S3Object": { "Bucket": "bucket",  
  "Name": "manifest_file" } } }'
```

Python

Use the following values to create a dataset. Supply the following command line parameters:

- `project_arn` — the ARN of the project that you want to add the test dataset to.
- `dataset_type` — the type of dataset that you want to create (train or test).
- `bucket` — the bucket that contains the manifest file for the dataset.
- `manifest_file` — the path and file name of the manifest file.

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.  
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-  
rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)
```

```
import boto3
import argparse
import logging
import time
import json
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def create_dataset(rek_client, project_arn, dataset_type, bucket, manifest_file):
    """
    Creates an Amazon Rekognition Custom Labels dataset.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_arn: The ARN of the project in which you want to create a
    dataset.
    :param dataset_type: The type of the dataset that you want to create (train or
    test).
    :param bucket: The S3 bucket that contains the manifest file.
    :param manifest_file: The path and filename of the manifest file.
    """

    try:
        #Create the project
        logger.info(f"Creating {dataset_type} dataset for project {project_arn}")

        dataset_type = dataset_type.upper()

        dataset_source = json.loads(
            '{ "GroundTruthManifest": { "S3Object": { "Bucket": "'
            + bucket
            + '", "Name": "'
            + manifest_file
            + '" } } }'
        )

        response = rek_client.create_dataset(
            ProjectArn=project_arn, DatasetType=dataset_type,
            DatasetSource=dataset_source
        )

        dataset_arn=response['DatasetArn']

        logger.info(f"dataset ARN: {dataset_arn}")

        finished=False
        while finished==False:

            dataset=rek_client.describe_dataset(DatasetArn=dataset_arn)

            status=dataset['DatasetDescription']['Status']

            if status == "CREATE_IN_PROGRESS":

                logger.info((f"Creating dataset: {dataset_arn} "))
                time.sleep(5)
                continue

            if status == "CREATE_COMPLETE":
                logger.info(f"Dataset created: {dataset_arn}")
                finished=True
                continue

            if status == "CREATE_FAILED":
                logger.exception(f"Dataset creation failed: {status} : {dataset_arn}")

    
```

```
        raise Exception(f"Dataset creation failed: {status} : {dataset_arn}")

        logger.exception(f"Failed. Unexpected state for dataset creation: {status} : {dataset_arn}")
        raise Exception(f"Failed. Unexpected state for dataset creation: {status} : {dataset_arn}")

    return dataset_arn

except ClientError as err:
    logger.exception(f"Couldn't create dataset: {err.response['Error']['Message']}")
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project in which you want to create the dataset."
    )

    parser.add_argument(
        "dataset_type", help="The type of the dataset that you want to create (train or test)."
    )

    parser.add_argument(
        "bucket", help="The S3 bucket that contains the manifest file."
    )

    parser.add_argument(
        "manifest_file", help="The path and filename of the manifest file."
    )

def main():
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    try:
        #get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Creating {args.dataset_type} dataset for project {args.project_arn}")

        #Create the project
        rek_client=boto3.client('rekognition')

        dataset_arn=create_dataset(rek_client,
                                   args.project_arn,
                                   args.dataset_type,
                                   args.bucket,
                                   args.manifest_file)

        print(f"Finished creating dataset: {dataset_arn}")

    except ClientError as err:
        logger.exception(f"Couldn't create dataset: {err.response['Error']['Message']}")
        raise
```

```
except ClientError as err:
    logger.exception(f"Problem creating dataset: {err}")
    print(f"Problem creating dataset: {err}")
except Exception as err:
    logger.exception(f"Problem creating dataset: {err}")
    print(f"Problem creating dataset: {err}")

if __name__ == "__main__":
    main()
```

Java V2

Use the following values to create a dataset. Supply the following command line parameters:

- `project_arn` — the ARN of the project that you want to add the test dataset to.
- `dataset_type` — the type of dataset that you want to create (train or test).
- `bucket` — the bucket that contains the manifest file for the dataset.
- `manifest_file` — the path and file name of the manifest file.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.CreateDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.CreateDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.DatasetDescription;
import software.amazon.awssdk.services.rekognition.model.DatasetSource;
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;
import software.amazon.awssdk.services.rekognition.model.DatasetType;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.GroundTruthManifest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.S3Object;

import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;

public class CreateDatasetManifestFiles {

    public static final Logger logger =
    Logger.getLogger(CreateDatasetManifestFiles.class.getName());

    public static String createMyDataset(RekognitionClient rekClient, String
    projectArn, String datasetType,
    String bucket, String name) throws Exception, RekognitionException {

        try {

            logger.log(Level.INFO, "Creating {0} dataset for project : {1} from
            s3://{2}/{3} ",
            new Object[] { datasetType.toString(), projectArn, bucket,
            name });

            DatasetType requestDatasetType = null;
```

```
switch (datasetType) {
    case "train":
        requestDatasetType = DatasetType.TRAIN;
        break;
    case "test":
        requestDatasetType = DatasetType.TEST;
        break;
    default:
        logger.log(Level.SEVERE, "Couldn't create dataset. Unrecognized
dataset type: {0}", datasetType);
        throw new Exception("Couldn't create dataset. Unrecognized dataset
type: " + datasetType);
}

GroundTruthManifest groundTruthManifest = GroundTruthManifest.builder()
.s3Object(S3Object.builder().bucket(bucket).name(name).build()).build();

DatasetSource datasetSource =
DatasetSource.builder().groundTruthManifest(groundTruthManifest).build();

CreateDatasetRequest createDatasetRequest =
CreateDatasetRequest.builder().projectArn(projectArn)

.datasetType(requestDatasetType).datasetSource(datasetSource).build();

CreateDatasetResponse response =
rekClient.createDataset(createDatasetRequest);

boolean created = false;

do {

    DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder()
    .datasetArn(response.datasetArn()).build();
    DescribeDatasetResponse describeDatasetResponse =
rekClient.describeDataset(describeDatasetRequest);

    DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

    DatasetStatus status = datasetDescription.status();

    logger.log(Level.INFO, "Creating dataset ARN: {0} ",
response.datasetArn());

    switch (status) {

        case CREATE_COMPLETE:
            logger.log(Level.INFO, "Dataset created");
            created = true;
            break;

        case CREATE_IN_PROGRESS:
            Thread.sleep(5000);
            break;

        case CREATE_FAILED:
            String error = "Dataset creation failed: " +
datasetDescription.statusAsString() + " "
                + datasetDescription.statusMessage() + " " +
response.datasetArn();
            logger.log(Level.SEVERE, error);
            throw new Exception(error);

    }
}
}
```

```
        default:
            String unexpectedError = "Unexpected creation state: " +
datasetDescription.statusAsString() + " "
+ datasetDescription.statusMessage() + " " +
response.datasetArn();
            logger.log(Level.SEVERE, unexpectedError);
            throw new Exception(unexpectedError);
        }

    } while (created == false);

    return response.datasetArn();

} catch (RekognitionException e) {
    logger.log(Level.SEVERE, "Could not create dataset: {}", e.getMessage());
    throw e;
}

}

public static void main(String args[]) {

    String datasetType = null;
    String bucket = null;
    String name = null;
    String projectArn = null;
    String datasetArn = null;

    final String USAGE = "\n" + "Usage: " + "<project_arn> <dataset_type>\n<dataset_arn>\n\n" + "Where:\n"
+ "    project_arn - the ARN of the project that you want to add
copy the dataset to.\n\n"
+ "    dataset_type - the type of the dataset that you want to
create (train or test).\n\n"
+ "    bucket - the S3 bucket that contains the manifest file.\n\n"
+ "    name - the location and name of the manifest file within the
bucket.\n\n";
}

if (args.length != 4) {
    System.out.println(USAGE);
    System.exit(1);
}

projectArn = args[0];
datasetType = args[1];
bucket = args[2];
name = args[3];

try {

    // Get the Rekognition client
    RekognitionClient rekClient = RekognitionClient.builder().build();

    // Create the dataset
    datasetArn = createMyDataset(rekClient, projectArn, datasetType,
bucket, name);

    System.out.println(String.format("Created dataset: %s", datasetArn));

    rekClient.close();

} catch (RekognitionException rekError) {
    logger.log(Level.SEVERE, "Rekognition client error: {}", rekError.getMessage());
}
```

```
        System.exit(1);
    } catch (Exception rekError) {
        logger.log(Level.SEVERE, "Error: {}", rekError.getMessage());
        System.exit(1);
    }
}
```

Creating a dataset using an existing dataset (SDK)

The following procedure shows you how to create a dataset from an existing dataset by using the [CreateDataset](#) operation.

- Use the following example code to create a dataset by copying another dataset.

AWS CLI

Use the following code to create the dataset. Replace the following:

- `project_arn` — the ARN of the project that you want to add the dataset to.
- `dataset_type` — with the type of dataset (TRAIN or TEST) that you want to create in the project.
- `dataset_arn` — with the ARN of the dataset that you want to copy.

```
aws rekognition create-dataset --project-arn project_arn \
--dataset-type dataset_type \
--dataset-source '{ "DatasetArn" : "dataset_arn" }'
```

Python

The following example creates a dataset using an existing dataset and displays its ARN.

To run the program, supply the following command line arguments:

- `project_arn` — the ARN of the project that you want to use.
- `dataset_type` — the type of the project dataset you want to create (train or test).
- `dataset_arn` — the ARN of the dataset that you want to create the dataset from.

```
# Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
# amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import argparse
import logging
import time
import json
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def create_dataset_from_existing_dataset(rek_client, project_arn, dataset_type,
    dataset_arn):
```

```
"""
Creates an Amazon Rekognition Custom Labels dataset using an existing dataset.
:param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
:param project_arn: The ARN of the project in which you want to create a
dataset.
:param dataset_type: The type of the dataset that you want to create (train or
test).
:param dataset_arn: The ARN of the existing dataset that you want to use.
"""

try:
    # Create the dataset

    dataset_type=dataset_type.upper()

    logger.info(
        f"Creating {dataset_type} dataset for project {project_arn} from
dataset {dataset_arn}.")

    dataset_source = json.loads(
        '{ "DatasetArn": "' + dataset_arn + '"}'
    )

    response = rek_client.create_dataset(
        ProjectArn=project_arn, DatasetType=dataset_type,
        DatasetSource=dataset_source
    )

    dataset_arn = response['DatasetArn']

    logger.info(f"New dataset ARN: {dataset_arn}")

    finished = False
    while finished == False:

        dataset = rek_client.describe_dataset(DatasetArn=dataset_arn)

        status = dataset['DatasetDescription']['Status']

        if status == "CREATE_IN_PROGRESS":

            logger.info((f"Creating dataset: {dataset_arn} "))
            time.sleep(5)
            continue

        if status == "CREATE_COMPLETE":
            logger.info(f"Dataset created: {dataset_arn}")
            finished = True
            continue

        if status == "CREATE_FAILED":
            logger.exception(
                f"Dataset creation failed: {status} : {dataset_arn}")
            raise Exception(
                f"Dataset creation failed: {status} : {dataset_arn}")

            logger.exception(
                f"Failed. Unexpected state for dataset creation: {status} :
{dataset_arn}")
            raise Exception(
                f"Failed. Unexpected state for dataset creation: {status} :
{dataset_arn}")

    return dataset_arn

except ClientError as err:
```

```
logger.exception(
    f"Couldn't create dataset: {err.response['Error']['Message']}")
raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """
    parser.add_argument(
        "project_arn", help="The ARN of the project in which you want to create the
dataset.")
    parser.add_argument(
        "dataset_type", help="The type of the dataset that you want to create
(train or test).")
    parser.add_argument(
        "dataset_arn", help="The ARN of the dataset that you want to copy from.")

def main():
    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")
    try:
        # get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(
            f"Creating {args.dataset_type} dataset for project {args.project_arn}")

        # Create the dataset
        rek_client=boto3.client('rekognition')

        dataset_arn = create_dataset_from_existing_dataset(rek_client,
                                                          args.project_arn,
                                                          args.dataset_type,
                                                          args.dataset_arn)

        print(f"Finished creating dataset: {dataset_arn}")

    except ClientError as err:
        logger.exception(f"Problem creating dataset: {err}")
        print(f"Problem creating dataset: {err}")
    except Exception as err:
        logger.exception(f"Problem creating dataset: {err}")
        print(f"Problem creating dataset: {err}")

if __name__ == "__main__":
    main()
```

Java V2

The following example creates a dataset using an existing dataset and displays its ARN.

To run the program, supply the following command line arguments:

- `project_arn` — the ARN of the project that you want to use.
- `dataset_type` — the type of the project dataset you want to create (`train` or `test`).
- `dataset_arn` — the ARN of the dataset that you want to create the dataset from.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.CreateDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.CreateDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.DatasetDescription;
import software.amazon.awssdk.services.rekognition.model.DatasetSource;
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;
import software.amazon.awssdk.services.rekognition.model.DatasetType;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;

public class CreateDatasetExisting {

    public static final Logger logger =
    Logger.getLogger(CreateDatasetExisting.class.getName());

    public static String createMyDataset(RekognitionClient rekClient, String
    projectArn, String datasetType,
    String existingDatasetArn) throws Exception, RekognitionException {

        try {

            logger.log(Level.INFO, "Creating {0} dataset for project : {1} from
dataset {2} ",
                       new Object[] { datasetType.toString(), projectArn,
existingDatasetArn });

            DatasetType requestDatasetType = null;

            switch (datasetType) {
                case "train":
                    requestDatasetType = DatasetType.TRAIN;
                    break;
                case "test":
                    requestDatasetType = DatasetType.TEST;
                    break;
                default:
                    logger.log(Level.SEVERE, "Unrecognized dataset type: {0}",
datasetType);
                    throw new Exception("Unrecognized dataset type: " + datasetType);
            }

            DatasetSource datasetSource =
DatasetSource.builder().datasetArn(existingDatasetArn).build();

            CreateDatasetRequest createDatasetRequest =
CreateDatasetRequest.builder().projectArn(projectArn)
```

```
.datasetType(requestDatasetType).datasetSource(datasetSource).build();

        CreateDatasetResponse response =
rekClient.createDataset(createDatasetRequest);

        boolean created = false;

        //Wait until create finishes

        do {

            DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder()
                .datasetArn(response.datasetArn()).build();
            DescribeDatasetResponse describeDatasetResponse =
rekClient.describeDataset(describeDatasetRequest);

            DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

            DatasetStatus status = datasetDescription.status();

            logger.log(Level.INFO, "Creating dataset ARN: {0} ",
response.datasetArn());

            switch (status) {

                case CREATE_COMPLETE:
                    logger.log(Level.INFO, "Dataset created");
                    created = true;
                    break;

                case CREATE_IN_PROGRESS:
                    Thread.sleep(5000);
                    break;

                case CREATE_FAILED:
                    String error = "Dataset creation failed: " +
datasetDescription.statusAsString() + " "
                        + datasetDescription.statusMessage() + " " +
response.datasetArn();
                    logger.log(Level.SEVERE, error);
                    throw new Exception(error);

                default:
                    String unexpectedError = "Unexpected creation state: " +
datasetDescription.statusAsString() + " "
                        + datasetDescription.statusMessage() + " " +
response.datasetArn();
                    logger.log(Level.SEVERE, unexpectedError);
                    throw new Exception(unexpectedError);
            }

        } while (created == false);

        return response.datasetArn();

    } catch (RekognitionException e) {
        logger.log(Level.SEVERE, "Could not create dataset: {0}",
e.getMessage());
        throw e;
    }
}
```

```
public static void main(String args[]) {  
    String datasetType = null;  
    String datasetArn = null;  
    String projectArn = null;  
    String datasetSourceArn = null;  
  
    final String USAGE = "\n" + "Usage: " + "<project_arn> <dataset_type>  
<dataset_arn>\n\n" + "Where:\n"  
        + "    project_arn - the ARN of the project that you want to add  
        copy the dataset to.\n\n"  
        + "    dataset_type - the type of the dataset that you want to  
        create (train or test).\n\n"  
        + "    dataset_arn - the ARN of the dataset that you want to copy  
        from.\n\n";  
  
    if (args.length != 3) {  
        System.out.println(USAGE);  
        System.exit(1);  
    }  
  
    projectArn = args[0];  
    datasetType = args[1];  
    datasetSourceArn = args[2];  
  
    try {  
  
        // Get the Rekognition client  
        RekognitionClient rekClient = RekognitionClient.builder().build();  
  
        // Create the dataset  
        datasetArn = createMyDataset(rekClient, projectArn, datasetType,  
        datasetSourceArn);  
  
        System.out.println(String.format("Created dataset: %s", datasetArn));  
  
        rekClient.close();  
  
    } catch (RekognitionException rekError) {  
        logger.log(Level.SEVERE, "Rekognition client error: {0}",  
        rekError.getMessage());  
        System.exit(1);  
    } catch (Exception rekError) {  
        logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());  
        System.exit(1);  
    }  
}  
}
```

Creating an empty dataset (SDK)

You can use the following code to create an empty training or test dataset with [CreateDataset](#). Later, add dataset entries by calling [UpdateDatasetEntries](#). For example code, see [Adding more images \(SDK\) \(p. 83\)](#).

To create an empty dataset (SDK)

1. If you haven't already:

- a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following examples to add JSON lines to a dataset.

CLI

Replace `project_arn` with the project that you want to add the dataset set to. Replace `dataset_type` with `TRAIN` to create a training dataset, or `TEST` to create a test dataset. to escape any special characters within the JSON Line.

```
aws rekognition create-dataset --project-arn "project_arn" \  
  --dataset-type dataset_type
```

Python

Use the following code to create a dataset. Supply the following command line options:

- `project_arn` — the ARN of the project that you want to add the test dataset to.
- `type` — the type of dataset that you want to create (train or test)

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.  
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)  
  
import boto3  
import argparse  
import logging  
import time  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)  
  
def create_empty_dataset(rek_client, project_arn, dataset_type):  
    """  
        Creates an empty Amazon Rekognition Custom Labels dataset.  
        :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.  
        :param project_arn: The ARN of the project in which you want to create a dataset.  
        :param dataset_type: The type of the dataset that you want to create (train or test).  
    """  
  
    try:  
        #Create the dataset  
        logger.info(f"Creating empty {dataset_type} dataset for project {project_arn}")  
  
        dataset_type=dataset_type.upper()  
  
        response = rek_client.create_dataset(  
            ProjectArn=project_arn, DatasetType=dataset_type  
        )  
  
        dataset_arn=response['DatasetArn']  
        logger.info(f"dataset ARN: {dataset_arn}")
```

```
finished=False
while finished==False:

    dataset=rek_client.describe_dataset(DatasetArn=dataset_arn)

    status=dataset['DatasetDescription']['Status']

    if status == "CREATE_IN_PROGRESS":

        logger.info((f"Creating dataset: {dataset_arn} "))
        time.sleep(5)
        continue

    if status == "CREATE_COMPLETE":
        logger.info(f"Dataset created: {dataset_arn}")
        finished=True
        continue

    if status == "CREATE_FAILED":
        logger.exception(f"Dataset creation failed: {status} : {dataset_arn}")
        raise Exception (f"Dataset creation failed: {status} : {dataset_arn}")

    logger.exception(f"Failed. Unexpected state for dataset creation: {status} : {dataset_arn}")
    raise Exception(f"Failed. Unexpected state for dataset creation: {status} : {dataset_arn}")

    return dataset_arn

except ClientError as err:
    logger.exception(f"Couldn't create dataset: {err.response['Error']['Message']}")
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project in which you want to create the empty dataset."
    )

    parser.add_argument(
        "dataset_type", help="The type of the empty dataset that you want to create (train or test)."
    )

def main():

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    try:

        #get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()
```

```
    print(f"Creating empty {args.dataset_type} dataset for project\n{args.project_arn}")

    #Create the empty dataset
    rek_client=boto3.client('rekognition')

    dataset_arn=create_empty_dataset(rek_client,
        args.project_arn,
        args.dataset_type.lower())

    print(f"Finished creating empty dataset: {dataset_arn}")

except ClientError as err:
    logger.exception(f"Problem creating empty dataset: {err}")
    print(f"Problem creating empty dataset: {err}")
except Exception as err:
    logger.exception(f"Problem creating empty dataset: {err}")
    print(f"Problem creating empty dataset: {err}")

if __name__ == "__main__":
    main()
```

Java 2

Use the following code to create a dataset. Supply the following command line options:

- `project_arn` — the ARN of the project that you want to add the test dataset to.
- `type` — the type of dataset that you want to create (train or test)

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.CreateDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.CreateDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.DatasetDescription;
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;
import software.amazon.awssdk.services.rekognition.model.DatasetType;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;

public class CreateEmptyDataset {

    public static final Logger logger =
    Logger.getLogger(CreateEmptyDataset.class.getName());

    public static String createMyEmptyDataset(RekognitionClient rekClient, String
    projectArn, String datasetType)
        throws Exception, RekognitionException {

        try {

            logger.log(Level.INFO, "Creating empty {0} dataset for project : {1}",
            new Object[] { datasetType.toString(), projectArn });

            CreateDatasetRequest createDatasetRequest =
            CreateDatasetRequest.builder()
                .datasetName(datasetType)
                .projectArn(projectArn)
                .build();

            CreateDatasetResponse createDatasetResponse =
            rekClient.createDataset(createDatasetRequest);

            DatasetDescription datasetDescription =
            createDatasetResponse.datasetDescription();

            logger.log(Level.INFO, "Dataset created successfully. Dataset ARN: {0}",
            datasetDescription.datasetArn());
```

```
DatasetType requestDatasetType = null;

switch (datasetType) {
    case "train":
        requestDatasetType = DatasetType.TRAIN;
        break;
    case "test":
        requestDatasetType = DatasetType.TEST;
        break;
    default:
        logger.log(Level.SEVERE, "Unrecognized dataset type: {0}",
datasetType);
        throw new Exception("Unrecognized dataset type: " + datasetType);
}

CreateDatasetRequest createDatasetRequest =
CreateDatasetRequest.builder().projectArn(projectArn)
    .datasetType(requestDatasetType).build();

CreateDatasetResponse response =
rekClient.createDataset(createDatasetRequest);

boolean created = false;

//Wait until updates finishes

do {

    DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder()
    .datasetArn(response.datasetArn()).build();
    DescribeDatasetResponse describeDatasetResponse =
rekClient.describeDataset(describeDatasetRequest);

    DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

    DatasetStatus status = datasetDescription.status();

    logger.log(Level.INFO, "Creating dataset ARN: {0} ",
response.datasetArn());

    switch (status) {

        case CREATE_COMPLETE:
            logger.log(Level.INFO, "Dataset created");
            created = true;
            break;

        case CREATE_IN_PROGRESS:
            Thread.sleep(5000);
            break;

        case CREATE_FAILED:
            String error = "Dataset creation failed: " +
datasetDescription.statusAsString() + " "
                + datasetDescription.statusMessage() + " " +
response.datasetArn();
            logger.log(Level.SEVERE, error);
            throw new Exception(error);

        default:
            String unexpectedError = "Unexpected creation state: " +
datasetDescription.statusAsString() + " "
    }
}
```

```
        + datasetDescription.statusMessage() + " " +
response.datasetArn();
        logger.log(Level.SEVERE, unexpectedError);
        throw new Exception(unexpectedError);
    }

} while (created == false);

return response.datasetArn();

} catch (RekognitionException e) {
    logger.log(Level.SEVERE, "Could not create dataset: {}", e.getMessage());
    throw e;
}

}

public static void main(String args[]) {

    String datasetType = null;
    String datasetArn = null;
    String projectArn = null;

    final String USAGE = "\n" + "Usage: " + "<project_arn> <dataset_type>\n\n" +
"Where:\n" +
"    + project_arn - the ARN of the project that you want to add
copy the dataset to.\n\n" +
"    + dataset_type - the type of the empty dataset that you want to
create (train or test).\n\n";

    if (args.length != 2) {
        System.out.println(USAGE);
        System.exit(1);
    }

    projectArn = args[0];
    datasetType = args[1];

    try {

        // Get the Rekognition client
        RekognitionClient rekClient = RekognitionClient.builder().build();

        // Create the dataset
        datasetArn = createMyEmptyDataset(rekClient, projectArn, datasetType);

        System.out.println(String.format("Created dataset: %s", datasetArn));

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {}", rekError.getMessage());
        System.exit(1);
    } catch (Exception rekError) {
        logger.log(Level.SEVERE, "Error: {}", rekError.getMessage());
        System.exit(1);
    }
}

}
```

}

3. Add images to the dataset. For more information, see [Adding more images \(SDK\) \(p. 218\)](#).

Adding more images (SDK)

`UpdateDatasetEntries` updates existing entries or adds new entries to a dataset. Each update is represented by a JSON line. You pass the JSON lines as a byte64 encoded data object in the `GroundTruth` field. If you are using an AWS SDK to call `UpdateDatasetEntries`, the SDK encodes the data for you. Each JSON line contains information for a single image, such as assigned labels or bounding box information. For example:

```
{"source-ref": "s3://bucket/image", "BB": {"annotations": [{"left": 1849, "top": 1039, "width": 422, "height": 283, "class_id": 0}, {"left": 1849, "top": 1340, "width": 443, "height": 415, "class_id": 1}, {"left": 2637, "top": 1380, "width": 676, "height": 338, "class_id": 2}, {"left": 2634, "top": 1051, "width": 673, "height": 338, "class_id": 3}], "image_size": [{"width": 4000, "height": 2667, "depth": 3}], "BB-metadata": {"job-name": "labeling-job/BB", "class-map": {"0": "comparator", "1": "pot_resistor", "2": "ir_phototransistor", "3": "ir_led"}, "human-annotated": "yes", "objects": [{"confidence": 1}, {"confidence": 1}, {"confidence": 1}, {"confidence": 1}], "creation-date": "2021-06-22T10:11:18.006Z", "type": "groundtruth/object-detection"}}
```

For more information, see [Creating a manifest file \(p. 242\)](#).

Use `source-ref` field as a key to identify images that you want to update. If the dataset doesn't contain a matching `source-ref` field value, the JSON line is added as a new image.

To add more images to a dataset (SDK)

1. If you haven't already:
 - Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following examples to add JSON lines to a dataset.

CLI

Replace the value of `GroundTruth` with the JSON Lines that you want to use. You need to escape any special characters within the JSON Line.

```
aws rekognition update-dataset-entries \
--dataset-arn dataset_arn \
--changes '{"GroundTruth": {"source-ref": "s3://your_bucket/your_image", "BB": {"annotations": [{"left": 1776, "top": 1017, "width": 458, "height": 317, "class_id": 0}, {"left": 1797, "top": 1334, "width": 418, "height": 415, "class_id": 1}, {"left": 2597, "top": 1361, "width": 655, "height": 329, "class_id": 2}, {"left": 2581, "top": 1020, "width": 689, "height": 338, "class_id": 3}], "image_size": [{"width": 4000, "height": 2667, "depth": 3}], "BB-metadata": {"job-name": "labeling-job/BB", "class-map": {"0": "comparator", "1": "pot_resistor", "2": "ir_phototransistor", "3": "ir_led"}, "human-annotated": "yes", "objects": [{"confidence": 1}, {"confidence": 1}, {"confidence": 1}, {"confidence": 1}], "creation-date": "2021-06-22T10:10:48.492Z", "type": "groundtruth/object-detection"}}}
```

Python

Use the following code. Supply the following command line parameters:

- `dataset_arn`— the ARN of the dataset that you want to update.
- `updates_file`— the file that contains the JSON Line updates.

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import argparse
import logging
import time
import json

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def update_dataset_entries(rek_client, dataset_arn, updates_file):
    """
    Adds dataset entries to an Amazon Rekognition Custom Labels dataset.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param dataset_arn: The ARN of the dataset that you want to update.
    :param updates_file: The manifest file of JSON Lines that contains the
    updates.
    """

    try:
        status=""
        status_message=""

        #Update dataset entries
        logger.info(f"Updating dataset {dataset_arn}")

        with open(updates_file) as f:
            manifest_file = f.read()

            changes=json.loads('{"GroundTruth" : ' +
                json.dumps(manifest_file) +
                '}')

            rek_client.update_dataset_entries(
                Changes=changes, DatasetArn=dataset_arn
            )

        finished=False
        while finished==False:

            dataset=rek_client.describe_dataset(DatasetArn=dataset_arn)

            status=dataset['DatasetDescription']['Status']
            status_message=dataset['DatasetDescription']['StatusMessage']

            if status == "UPDATE_IN_PROGRESS":

                logger.info((f"Updating dataset: {dataset_arn} "))
                time.sleep(5)

    except ClientError as error:
        logger.error(error)

    return status
```

```
        continue

        if status == "UPDATE_COMPLETE":
            logger.info(f"Dataset updated: {status} : {status_message} :
{dataset_arn}")
            finished=True
            continue

        if status == "UPDATE_FAILED":
            logger.exception(f"Dataset update failed: {status} :
{status_message} : {dataset_arn}")
            raise Exception (f"Dataset update failed: {status} :
{status_message} : {dataset_arn}")

            logger.exception(f"Failed. Unexpected state for dataset update:
{status} : {status_message} : {dataset_arn}")
            raise Exception(f"Failed. Unexpected state for dataset update:
{status} : {status_message} :{dataset_arn}")

        logger.info(f"Added entries to dataset")

        return status, status_message

    except ClientError as err:
        logger.exception(f"Couldn't update dataset: {err.response['Error'][
'Message']}"))
        raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "dataset_arn", help="The ARN of the dataset that you want to update."
    )

    parser.add_argument(
        "updates_file", help="The manifest file of JSON Lines that contains the
updates."
    )

def main():

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    try:

        #get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Updating dataset {args.dataset_arn} with entries from
{args.updates_file}.")

        #Update the dataset
        rek_client=boto3.client('rekognition')

        status, status_message=update_dataset_entries(rek_client,
            args.dataset_arn,
            args.updates_file)
```

```
        print(f"Finished updates dataset: {status} : {status_message}")

    except ClientError as err:
        logger.exception(f"Problem updating dataset: {err}")
        print(f"Problem updating dataset: {err}")
    except Exception as err:
        logger.exception(f"Problem updating dataset: {err}")
        print(f"Problem updating dataset: {err}")

if __name__ == "__main__":
    main()
```

Java 2

- `dataset_arn`— the ARN of the dataset that you want to update.
- `update_file`— the file that contains the JSON Line updates.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DatasetChanges;
import software.amazon.awssdk.services.rekognition.model.DatasetDescription;
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
software.amazon.awssdk.services.rekognition.model.UpdateDatasetEntriesRequest;
import
software.amazon.awssdk.services.rekognition.model.UpdateDatasetEntriesResponse;

import java.io.FileInputStream;
import java.io.InputStream;
import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;

public class UpdateDatasetEntries {

    public static final Logger logger =
    Logger.getLogger(UpdateDatasetEntries.class.getName());

    public static String updateMyDataset(RekognitionClient rekClient, String
datasetArn,
        String updateFile
    ) throws Exception, RekognitionException {

        try {

            logger.log(Level.INFO, "Updating dataset {0}",
            new Object[] { datasetArn});

            InputStream sourceStream = new FileInputStream(updateFile);
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

            DatasetChanges datasetChanges = DatasetChanges.builder()
```

```
        .groundTruth(sourceBytes).build();

        UpdateDatasetEntriesRequest updateDatasetEntriesRequest =
UpdateDatasetEntriesRequest.builder()
        .changes(datasetChanges)
        .datasetArn(datasetArn)
        .build();

        UpdateDatasetEntriesResponse response =
rekClient.updateDatasetEntries(updateDatasetEntriesRequest);

        boolean updated = false;

        //Wait until update completes

        do {

            DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder()
            .datasetArn(datasetArn).build();
            DescribeDatasetResponse describeDatasetResponse =
rekClient.describeDataset(describeDatasetRequest);

            DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

            DatasetStatus status = datasetDescription.status();

            logger.log(Level.INFO, " dataset ARN: {0} ", datasetArn);

            switch (status) {

                case UPDATE_COMPLETE:
                    logger.log(Level.INFO, "Dataset updated");
                    updated = true;
                    break;

                case UPDATE_IN_PROGRESS:
                    Thread.sleep(5000);
                    break;

                case UPDATE_FAILED:
                    String error = "Dataset update failed: " +
datasetDescription.statusAsString() + " "
                        + datasetDescription.statusMessage() + " " +
datasetArn;
                    logger.log(Level.SEVERE, error);
                    throw new Exception(error);

                default:
                    String unexpectedError = "Unexpected update state: " +
datasetDescription.statusAsString() + " "
                        + datasetDescription.statusMessage() + " " +
datasetArn;
                    logger.log(Level.SEVERE, unexpectedError);
                    throw new Exception(unexpectedError);
            }

        } while (updated == false);

        return datasetArn;

    } catch (RekognitionException e) {
        logger.log(Level.SEVERE, "Could not update dataset: {0}",
e.getMessage());
        throw e;
    }
}
```

```

        }

    }

    public static void main(String args[]) {
        String updatesFile = null;
        String datasetArn = null;

        final String USAGE = "\n" + "Usage: " + "<project_arn> <dataset_arn>\n" +
            "<updates_file>\n" + "Where:\n" +
            "    dataset_arn - the ARN of the dataset that you want to update.\n" +
            "\n" +
            "    update_file - The file that includes in JSON Line updates.\n" +
            "\n";

        if (args.length != 2) {
            System.out.println(USAGE);
            System.exit(1);
        }

        datasetArn = args[0];
        updatesFile = args[1];

        try {
            // Get the Rekognition client
            RekognitionClient rekClient = RekognitionClient.builder().build();

            // Update the dataset
            datasetArn = updateMyDataset(rekClient, datasetArn, updatesFile);

            System.out.println(String.format("Dataset updated: %s", datasetArn));

            rekClient.close();

        } catch (RekognitionException rekError) {
            logger.log(Level.SEVERE, "Rekognition client error: {0}", rekError.getMessage());
            System.exit(1);
        } catch (Exception rekError) {
            logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());
            System.exit(1);
        }
    }
}

```

Splitting a training dataset to create a test dataset (SDK)

Amazon Rekognition Custom Labels requires a training dataset and a test dataset to train your model.

If you are using the API, you can use the [DistributeDatasetEntries](#) operation to distribute 20% of the training dataset into an empty test dataset. Distributing the training dataset can be useful if you only have a single manifest file available. Use the single manifest file to create your training dataset. Then create an empty test dataset and use [DistributeDatasetEntries](#) to populate the test dataset.

Note

If you are using the Amazon Rekognition Custom Labels console and start with a single dataset project, Amazon Rekognition Custom Labels splits (distributes) the training dataset, during

training, to create a test dataset. 20% of the training dataset entries are moved to the test dataset.

To distribute a training dataset (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with AmazonRekognitionFullAccess permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
 - c. Create a project. For more information, see [Creating an Amazon Rekognition Custom Labels project \(SDK\) \(p. 51\)](#).
 - d. Create your training dataset. For more information, see [Training dataset \(p. 65\)](#).
2. Create an empty test dataset by following the instructions at [the section called "Creating an empty dataset \(SDK\)" \(p. 77\)](#).
3. Use the following example code to distribute 20% of the training dataset entries into the test dataset.

AWS CLI

Change the value of `training_dataset_arn` and `test_dataset_arn` with the ARNs of the datasets that you want to use.

```
aws rekognition distribute-dataset-entries --datasets ['{"Arn": "training_dataset_arn"}, {"Arn": "test_dataset_arn"}']
```

Python

Use the following code. Supply the following command line parameters:

- `training_dataset_arn` — the ARN of the training dataset that you distribute entries from.
- `test_dataset_arn` — the ARN of the test dataset that you distribute entries to.

```
# Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
# amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import argparse
import logging
import time
import json
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def check_dataset_status(rek_client, dataset_arn):
    finished = False
    status = ""
    status_message = ""

    while finished == False:
        dataset = rek_client.describe_dataset(DatasetArn=dataset_arn)
```

```
status = dataset['DatasetDescription']['Status']
status_message = dataset['DatasetDescription']['StatusMessage']

if status == "UPDATE_IN_PROGRESS":

    logger.info(f"Distributing dataset: {dataset_arn} ")
    time.sleep(5)
    continue

if status == "UPDATE_COMPLETE":
    logger.info(
        f"Dataset distribution complete: {status} : {status_message} :
{dataset_arn}")
    finished = True
    continue

if status == "UPDATE_FAILED":
    logger.exception(
        f"Dataset distribution failed: {status} : {status_message} :
{dataset_arn}")
    finished = True
    break

logger.exception(
    f"Failed. Unexpected state for dataset distribution: {status} :
{status_message} : {dataset_arn}")
finished = True
status_message = "An unexpected error occurred while distributing the
dataset"
break

return status, status_message

def distribute_dataset_entries(rek_client, training_dataset_arn, test_dataset_arn):
    """
    Distributes 20% of the supplied training dataset into the supplied test
    dataset.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param training_dataset_arn: The ARN of the training dataset that you
    distribute entries from.
    :param test_dataset_arn: The ARN of the test dataset that you distribute
    entries to.
    """

    try:
        # List dataset labels
        logger.info(f"Distributing training dataset entries
({training_dataset_arn}) into test dataset ({test_dataset_arn})."
)
        datasets = json.loads(
            '[{"Arn" : "' + str(training_dataset_arn) + '"}, {"Arn" : "' +
            str(test_dataset_arn) + '"}]')
        rek_client.distribute_dataset_entries(
            Datasets=datasets
        )

        training_dataset_status, training_dataset_status_message =
check_dataset_status(
            rek_client, training_dataset_arn)
        test_dataset_status, test_dataset_status_message = check_dataset_status(
            rek_client, test_dataset_arn)
    
```

```
        if training_dataset_status == 'UPDATE_COMPLETE' and test_dataset_status ==
"UPDATE_COMPLETE":
            print(f"Distribution complete")
        else:
            print("Distribution failed:")
            print(
                f"\ttraining dataset: {training_dataset_status} :
{training_dataset_status_message}")
            print(
                f"\ttest dataset: {test_dataset_status} :
{test_dataset_status_message}")

    except ClientError as err:
        logger.exception(
            f"Couldn't distribute dataset: {err.response['Error']['Message']}"))
        raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """
    parser.add_argument(
        "training_dataset_arn", help="The ARN of the training dataset that you want
        to distribute from."
    )

    parser.add_argument(
        "test_dataset_arn", help="The ARN of the test dataset that you want to
        distribute to."
    )

def main():
    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:
        # get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(
            f"Distributing training dataset entries ({args.training_dataset_arn})
            into test dataset ({args.test_dataset_arn}).")

        # Distribute the datasets

        rek_client = boto3.client('rekognition')

        distribute_dataset_entries(rek_client,
                                    args.training_dataset_arn,
                                    args.test_dataset_arn)

        print(f"Finished distributing datasets.")

    except ClientError as err:
        logger.exception(f"Problem distributing datasets: {err}")
        print(f"Problem listing dataset labels: {err}")
    except Exception as err:
        logger.exception(f"Problem distributing datasets: {err}")


```

```
        print(f"Problem distributing datasets: {err}")

if __name__ == "__main__":
    main()
```

Java 2

Use the following code. Supply the following command line parameters:

- `training_dataset_arn` — the ARN of the training dataset that you distribute entries from.
- `test_dataset_arn` — the ARN of the test dataset that you distribute entries to.

```
//Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DatasetDescription;
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.DistributeDataset;
import
software.amazon.awssdk.services.rekognition.model.DistributeDatasetEntriesRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

public class DistributeDatasetEntries {

    public static final Logger logger =
    Logger.getLogger(DistributeDatasetEntries.class.getName());

    public static DatasetStatus checkDatasetStatus(RekognitionClient rekClient,
    String datasetArn)
        throws Exception, RekognitionException {

        boolean distributed = false;
        DatasetStatus status = null;

        // Wait until distribution completes

        do {

            DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder().datasetArn(datasetArn)
            .build();
            DescribeDatasetResponse describeDatasetResponse =
rekClient.describeDataset(describeDatasetRequest);

            DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

            status = datasetDescription.status();

            logger.log(Level.INFO, " dataset ARN: {0} ", datasetArn);

            switch (status) {

                case UPDATE_COMPLETE:
```

```
        logger.log(Level.INFO, "Dataset updated");
        distributed = true;
        break;

    case UPDATE_IN_PROGRESS:
        Thread.sleep(5000);
        break;

    case UPDATE_FAILED:
        String error = "Dataset distribution failed: " +
datasetDescription.statusAsString() + " "
            + datasetDescription.statusMessage() + " " + datasetArn;
        logger.log(Level.SEVERE, error);
        break;

    default:
        String unexpectedError = "Unexpected distribution state: " +
datasetDescription.statusAsString() + " "
            + datasetDescription.statusMessage() + " " + datasetArn;
        logger.log(Level.SEVERE, unexpectedError);

    }

} while (distributed == false);

return status;
}

public static void distributeMyDatasetEntries(RekognitionClient rekClient,
String trainingDatasetArn,
String testDatasetArn) throws Exception, RekognitionException {
try {

    logger.log(Level.INFO, "Distributing {0} dataset to {1} ",
        new Object[] { trainingDatasetArn, testDatasetArn });

    DistributeDataset distributeTrainingDataset =
DistributeDataset.builder().arn(trainingDatasetArn).build();

    DistributeDataset distributeTestDataset =
DistributeDataset.builder().arn(testDatasetArn).build();

    ArrayList<DistributeDataset> datasets = new ArrayList();

    datasets.add(distributeTrainingDataset);
    datasets.add(distributeTestDataset);

    DistributeDatasetEntriesRequest distributeDatasetEntriesRequest =
DistributeDatasetEntriesRequest.builder()
        .datasets(datasets).build();

    rekClient.distributeDatasetEntries(distributeDatasetEntriesRequest);

    DatasetStatus trainingStatus = checkDatasetStatus(rekClient,
trainingDatasetArn);
    DatasetStatus testStatus = checkDatasetStatus(rekClient,
testDatasetArn);

    if (trainingStatus == DatasetStatus.UPDATE_COMPLETE && testStatus ==
DatasetStatus.UPDATE_COMPLETE) {
        logger.log(Level.INFO, "Successfully distributed dataset: {0}");
    } else {


```

```
        throw new Exception("Failed to distribute dataset: " +
trainingDatasetArn);
    }

} catch (RekognitionException e) {
    logger.log(Level.SEVERE, "Could not distribute dataset: {0}",
e.getMessage());
    throw e;
}

}

public static void main(String args[]) {

    String trainingDatasetArn = null;
    String testDatasetArn = null;

    final String USAGE = "\n" + "Usage: " + "<training_dataset_arn>
<test_dataset_arn>\n\n" + "Where:\n"
        + "    training_dataset_arn - the ARN of the dataset that you want
to distribute from.\n\n"
        + "    test_dataset_arn - the ARN of the dataset that you want to
distribute to.\n\n";

    if (args.length != 2) {
        System.out.println(USAGE);
        System.exit(1);
    }

    trainingDatasetArn = args[0];
    testDatasetArn = args[1];

    try {

        // Get the Rekognition client
        RekognitionClient rekClient = RekognitionClient.builder().build();

        // Distribute the dataset
        distributeMyDatasetEntries(rekClient, trainingDatasetArn,
testDatasetArn);

        System.out.println("Datasets distributed.");

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    } catch (Exception rekError) {
        logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());
        System.exit(1);
    }

}

}
```

Debugging datasets

During dataset creation there are two types of error that can occur — *terminal errors* and *non-terminal errors*. Terminal errors can stop dataset creation or update. Non-terminal errors don't stop dataset creation or update.

Topics

- [Terminal errors \(p. 95\)](#)
- [Non-terminal errors \(p. 98\)](#)

Terminal errors

There are two types of terminal errors — file errors that cause dataset creation to fail, and content errors that Amazon Rekognition Custom Labels removes from the dataset. Dataset creation fails if there are too many content errors.

Topics

- [Terminal file errors \(p. 95\)](#)
- [Terminal content errors \(p. 98\)](#)

Terminal file errors

The following are file errors. You can get information about file errors by calling `DescribeDataset` and checking the `Status` and `StatusMessage` fields. For example code, see [Describing a dataset \(SDK\) \(p. 224\)](#).

- [ERROR_MANIFEST_INACCESSIBLE_OR_UNSUPPORTED_FORMAT \(p. 95\)](#)
- [ERROR_MANIFEST_SIZE_TOO_LARGE \(p. 96\)](#).
- [ERROR_MANIFEST_ROWS_EXCEEDS_MAXIMUM \(p. 96\)](#)
- [ERROR_INVALID_PERMISSIONS_MANIFEST_S3_BUCKET \(p. 96\)](#)
- [ERROR_TOO_MANY_RECORDS_IN_ERROR \(p. 97\)](#)
- [ERROR_MANIFEST_TOO_MANY_LABELS \(p. 97\)](#)
- [ERROR_INSUFFICIENT_IMAGES_PER_LABEL_FOR_DISTRIBUTE \(p. 97\)](#)

ERROR_MANIFEST_INACCESSIBLE_OR_UNSUPPORTED_FORMAT

Error message

The manifest file extension or contents are invalid.

The training or testing manifest file doesn't have a file extension or its contents are invalid.

To fix error `ERROR_MANIFEST_INACCESSIBLE_OR_UNSUPPORTED_FORMAT`

- Check the following possible causes in both the training and testing manifest files.
 - The manifest file is missing a file extension. By convention the file extension is `.manifest`.
 - The Amazon S3 bucket or key for the manifest file couldn't be found.

[ERROR_MANIFEST_SIZE_TOO_LARGE](#)

Error message

The manifest file size exceeds the maximum supported size.

The training or testing manifest file size (in bytes) is too large. For more information, see [Guidelines and quotas in Amazon Rekognition Custom Labels \(p. 323\)](#). A manifest file can have less than the maximum number of JSON Lines and still exceed the maximum file size.

You can't use the Amazon Rekognition Custom Labels console to fix error *The manifest file size exceeds the maximum supported size.*

To fix error [ERROR_MANIFEST_SIZE_TOO_LARGE](#)

1. Check which of the training and testing manifests exceed the maximum file size.
2. Reduce the number of JSON Lines in the manifest files that are too large. For more information, see [Creating a manifest file \(p. 242\)](#).

[ERROR_MANIFEST_ROWS_EXCEEDS_MAXIMUM](#)

Error message

The manifest file has too many rows.

More information

The number of JSON Lines (number of images) in the manifest file is greater than the allowed limit. The limit is different for image-level models and object location models. For more information, see [Guidelines and quotas in Amazon Rekognition Custom Labels \(p. 323\)](#).

JSON Line error are validated until the number of JSON Lines reaches the [ERROR_MANIFEST_ROWS_EXCEEDS_MAXIMUM](#) limit.

You can't use the Amazon Rekognition Custom Labels console to fix error [ERROR_MANIFEST_ROWS_EXCEEDS_MAXIMUM](#).

To fix [ERROR_MANIFEST_ROWS_EXCEEDS_MAXIMUM](#)

- Reduce the number of JSON Lines in the manifest. For more information, see [Creating a manifest file \(p. 242\)](#).

[ERROR_INVALID_PERMISSIONS_MANIFEST_S3_BUCKET](#)

Error message

The S3 bucket permissions are incorrect.

Amazon Rekognition Custom Labels doesn't have permissions to one or more of the buckets containing the training and testing manifest files.

You can't use the Amazon Rekognition Custom Labels console to fix this error.

To fix error [ERROR_INVALID_PERMISSIONS_MANIFEST_S3_BUCKET](#)

- Check the permissions for the bucket(s) containing the training and testing manifests. For more information, see [Step 4: Set up Amazon Rekognition Custom Labels permissions \(p. 6\)](#).

[ERROR_TOO_MANY_RECORDS_IN_ERROR](#)

Error message

The manifest file has too many terminal errors.

To fix [ERROR_TOO_MANY_RECORDS_IN_ERROR](#)

- Reduce the number of JSON Lines (images) with terminal content errors. For more information, see [Terminal manifest content errors \(p. 129\)](#).

You can't use the Amazon Rekognition Custom Labels console to fix this error.

[ERROR_MANIFEST_TOO_MANY_LABELS](#)

Error message

The manifest file has too many labels.

More information

The number of unique labels in the manifest (dataset) is more than the allowed limit. If the training dataset is split to create a testing dataset, the number of labels is determined after the split.

To fix [ERROR_MANIFEST_TOO_MANY_LABELS \(Console\)](#)

- Remove labels from the dataset. For more information, see [Managing labels \(p. 100\)](#). The labels are automatically removed from the images and bounding boxes in your dataset.

To fix [ERROR_MANIFEST_TOO_MANY_LABELS \(JSON Line\)](#)

- Manifests with image level JSON Lines – If the image has a single label, remove the JSON Lines for images that use the desired label. If the JSON Line contains multiple labels, remove only the JSON object for the desired label. For more information, see [Adding multiple image-level labels to an image \(p. 251\)](#).

Manifests with object location JSON Lines – Remove the bounding box and associated label information for the label that you want to remove. Do this for each JSON Line that contains the desired label. You need to remove the label from the `class-map` array and corresponding objects in the `objects` and `annotations` array. For more information, see [Object localization in manifest files \(p. 252\)](#).

[ERROR_INSUFFICIENT_IMAGES_PER_LABEL_FOR_DISTRIBUTE](#)

Error message

The manifest file doesn't have enough labeled images to distribute the dataset.

Dataset distribution occurs when Amazon Rekognition Custom Labels splits a training dataset to create a test dataset. You can also split a dataset by calling the `DistributeDatasetEntries` API.

To fix error [ERROR_MANIFEST_TOO_MANY_LABELS](#)

- Add more labeled images to the training dataset

Terminal content errors

The following are terminal content errors. During dataset creation, images that have terminal content errors are removed from the dataset. The dataset can still be used for training. If there are too many content errors, dataset/update fails. Terminal content errors related to dataset operations aren't displayed in the console or returned from `DescribeDataset` or other API. If you notice that images or annotations are missing from your datasets, check your dataset manifest files for the following issues:

- The length of a JSON line is too long. The maximum length is 100,000 characters.
- The `source-ref` value is missing from a JSON Line.
- The format of a `source-ref` value in a JSON Line is invalid.
- The contents of a JSON Line are not valid.
- The value a `source-ref` field appears more than once. An image can only be referenced once in a dataset.

For information about the `source-ref` field, see [Creating a manifest file \(p. 242\)](#).

Non-terminal errors

The following are non-terminal errors that can occur during dataset creation or update. These errors can invalidate an entire JSON Line or invalidate annotations within a JSON Line. If a JSON Line has an error, it is not used for training. If an annotation within a JSON Line has an error, the JSON Line is still used for training, but without the broken annotation. For more information about JSON Lines, see [Creating a manifest file \(p. 242\)](#).

You can access non-terminal errors from the console and by calling the `ListDatasetEntries` API. For more information, see [Listing dataset entries \(SDK\) \(p. 227\)](#).

The following errors are also returned during training. We recommend that you fix these errors before training your model. For more information, see [Non-Terminal JSON Line Validation Errors \(p. 135\)](#).

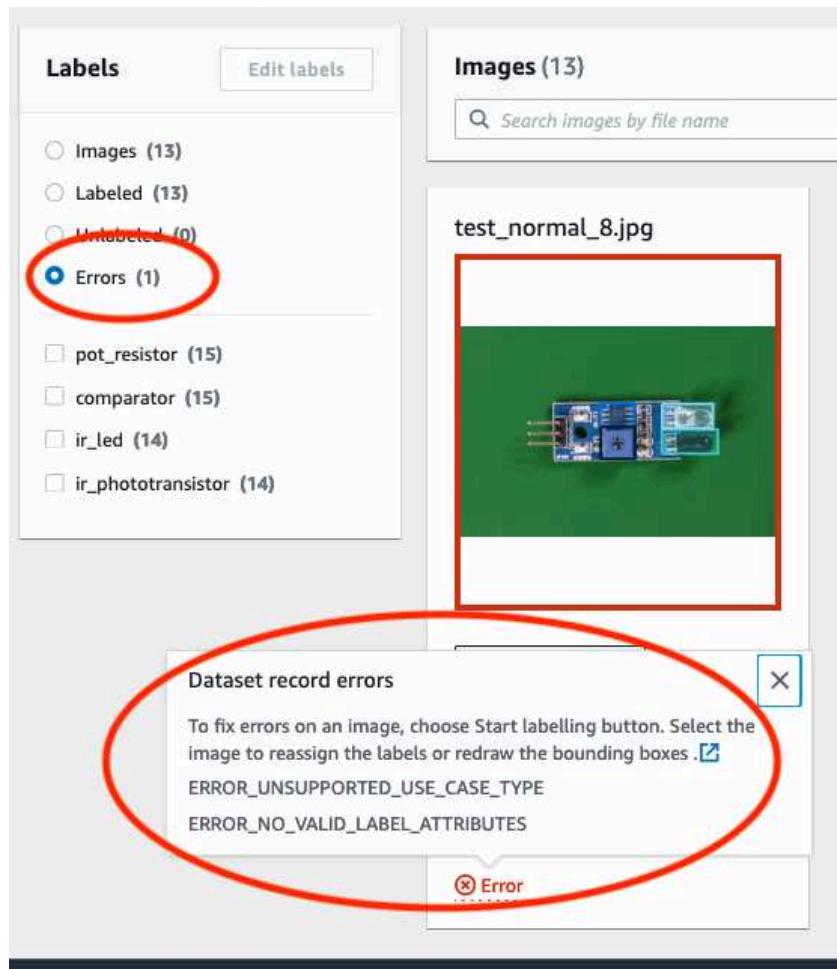
- [ERROR_NO_LABEL_ATTRIBUTES \(p. 136\)](#)
- [ERROR_INVALID_LABEL_ATTRIBUTE_FORMAT \(p. 137\)](#)
- [ERROR_INVALID_LABEL_ATTRIBUTE_METADATA_FORMAT \(p. 138\)](#)
- [ERROR_NO_VALID_LABEL_ATTRIBUTES \(p. 139\)](#)
- [ERROR_INVALID_BOUNDING_BOX \(p. 143\)](#)
- [ERROR_INVALID_IMAGE_DIMENSION \(p. 142\)](#)
- [ERROR_BOUNDING_BOX_TOO_SMALL \(p. 145\)](#)
- [ERROR_NO_VALID_ANNOTATIONS \(p. 144\)](#)
- [ERROR_MISSING_BOUNDING_BOX_CONFIDENCE \(p. 139\)](#)
- [ERROR_MISSING_CLASS_MAP_ID \(p. 140\)](#)
- [ERROR_TOO_MANY_BOUNDING_BOXES \(p. 145\)](#)
- [ERROR_UNSUPPORTED_USE_CASE_TYPE \(p. 149\)](#)
- [ERROR_INVALID_LABEL_NAME_LENGTH \(p. 151\)](#)

Accessing non-terminal errors

You can use the console to find out which images in a dataset have non-terminal errors. You can also call, call `ListDatasetEntries` API to get the error messages. For more information, see [Listing dataset entries \(SDK\) \(p. 227\)](#).

To access non-terminal errors(console)

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. Choose **Use Custom Labels**.
3. Choose **Get started**.
4. In the left navigation pane, choose **Projects**.
5. In the **Projects** page, choose the project that you want to use. The details page for your project is displayed.
6. If you want to view non-terminal errors in your training dataset, choose the **Training** tab. Otherwise choose the **Test** tab to view non-terminal errors in your test dataset.
7. In the **Labels** section of the dataset gallery, choose **Errors**. The dataset gallery is filtered to only show images with errors.
8. Choose **Error** underneath an image to see the error code. Use the information at [Non-Terminal JSON Line Validation Errors \(p. 135\)](#) to fix the error.



Labeling images

A label identifies an object, scene, concept, or bounding box around an object in an image. For example, if your dataset contains images of dogs, you might add labels for breeds of dogs.

After importing your images into a dataset, you might need to add labels to images or correct mislabeled images. For example, images aren't labeled if they are imported from a local computer. You use the dataset gallery to add new labels to the dataset and assign labels and bounding boxes to images in the dataset.

How you label the images in your datasets determines the type of model that Amazon Rekognition Custom Labels trains. For more information, see [Purposing datasets \(p. 55\)](#).

Topics

- [Managing labels \(p. 100\)](#)
- [Assigning image-level labels to an image \(p. 102\)](#)
- [Labeling objects with bounding boxes \(p. 104\)](#)

Managing labels

You can manage labels by using the Amazon Rekognition Custom Labels console. There isn't a specific API for managing labels – labels are added to the dataset when you create the dataset with `CreateDataset` or when you add more images to the dataset with `UpdateDatasetEntries`.

Topics

- [Managing labels \(Console\) \(p. 100\)](#)
- [Managing Labels \(SDK\) \(p. 101\)](#)

Managing labels (Console)

You can use the Amazon Rekognition Custom Labels console to add, change, or remove labels from a dataset. To add a label to a dataset, you can add a new label that you create or import labels from an existing dataset in Rekognition.

Topics

- [Add new labels \(Console\) \(p. 100\)](#)
- [Change and remove labels \(Console\) \(p. 101\)](#)

Add new labels (Console)

You can specify new labels that you want to add to your dataset.

Add labels using the editing window

To add a new label (console)

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. Choose **Use Custom Labels**.
3. Choose **Get started**.
4. In the left navigation pane, choose **Projects**.
5. In the **Projects** page, choose the project that you want to use. The details page for your project is displayed.
6. If you want to add labels to your training dataset, choose the **Training** tab. Otherwise choose the **Test** tab to add labels to the test dataset.
7. Choose **Start labeling** to enter labeling mode.
8. In the **Labels** section of the dataset gallery, choose **Manage labels** to open the **Manage labels** dialog box.
9. In the edit box, enter a new label name.

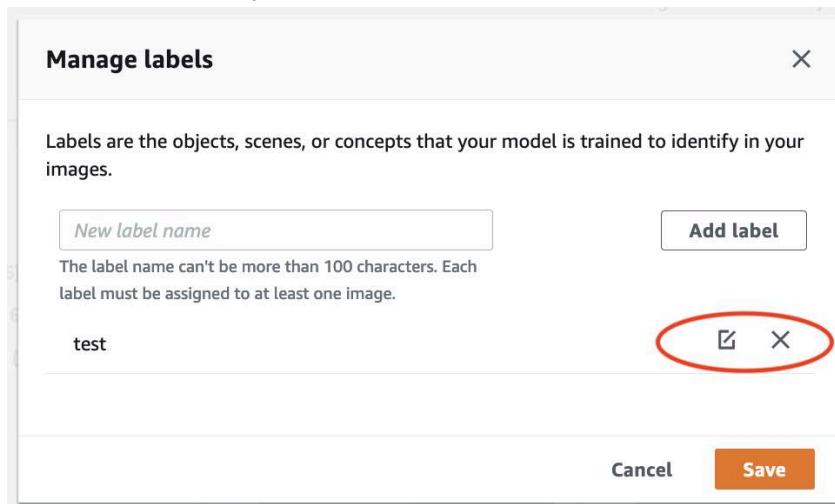
10. Choose **Add label**.
11. Repeat steps 9 and 10 until you have created all the labels you need.
12. Choose **Save** to save the labels that you added.

Change and remove labels (Console)

You can rename or remove labels after adding them to a dataset. You can only remove labels that are not assigned to any images.

To rename or remove an existing label (console)

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. Choose **Use Custom Labels**.
3. Choose **Get started**.
4. In the left navigation pane, choose **Projects**.
5. In the **Projects** page, choose the project that you want to use. The details page for your project is displayed.
6. If you want to change or delete labels in your training dataset, choose the **Training** tab. Otherwise choose the **Test** tab to change or delete labels to the test dataset.
7. Choose **Start labeling** to enter labeling mode.
8. In the **Labels** section of the dataset gallery, choose **Manage labels** to open the **Manage labels** dialog box.
9. Choose the label that you want to edit or delete.



- a. If you choose the delete icon (X), the label is removed from the list.
 - b. If you want to change the label, choose the edit icon (pencil and paper pad) and enter a new label name in the edit box.
10. Choose **Save** to save your changes.

Managing Labels (SDK)

There isn't a unique API that manages dataset labels. If you create a dataset with `CreateDataset`, the labels found in the manifest file or copied dataset, create the initial set of labels. If you add more images with the `UpdateDatasetEntries` API, new labels found in the entries are added to the dataset. For more information, see [Adding more images \(SDK\) \(p. 218\)](#). To delete labels from a dataset, you must remove all label annotations in the dataset.

To delete labels from a dataset

1. Call `ListDatasetEntries` to get the dataset entries. For example code, see [Listing dataset entries \(SDK\) \(p. 227\)](#).
2. In the file, remove any label annotations. For more information, see [Image-Level labels in manifest files \(p. 249\)](#) and the section called “Object localization in manifest files” (p. 252).
3. Use the file to update the dataset with the `UpdateDatasetEntries` API. For more information, see [Adding more images \(SDK\) \(p. 218\)](#).

Assigning image-level labels to an image

You use image-level labels to train models that classify images into categories. An image-level label indicates that an image contains an object, scene or concept. For example, the following image shows a river. If your model classifies images as containing rivers, you would add a *river* image-level label. For more information, see [Purposing datasets \(p. 55\)](#).

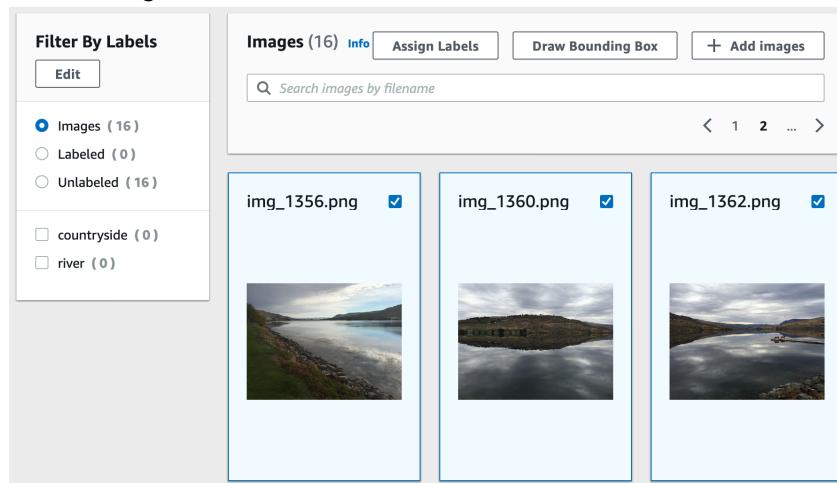


A dataset that contains image-level labels, needs at least two labels defined. Each image needs at least one assigned label that identifies the object, scene, or concept in the image.

To assign image-level labels to an image (console)

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. Choose **Use Custom Labels**.
3. Choose **Get started**.
4. In the left navigation pane, choose **Projects**.

5. In the **Projects** page, choose the project that you want to use. The details page for your project is displayed.
6. On the project details page, choose **Label images**
7. If you want to add labels to your training dataset, choose the **Training** tab. Otherwise choose the **Test** tab to add labels to the test dataset.
8. Choose **Start labeling** to enter labeling mode.
9. In the image gallery, select one or more images that you want to add labels to. You can only select images on a single page at a time. To select a contiguous range of images on a page:
 - a. Select the first image in the range.
 - b. Press and hold the shift key.
 - c. Select the last image range. The images between the first and second image are also selected.
 - d. Release the shift key.
10. Choose **Assign Labels**.



11. In **Assign a label to selected images** dialog box, select a label that you want to assign to the image or images.
12. Choose **Assign image-level labels** to assign label to the image.
13. Repeat labeling until every image is annotated with the required labels.
14. Choose **Save changes** to save your changes.
15. Choose **Exit** to exit labeling mode.

Assign image-level labels (SDK)

You can use the `UpdateDatasetEntries` API to add or update the image-level labels that are assigned to an image. `UpdateDatasetEntries` takes one or more JSON lines. Each JSON Line represents a single image. For an image with an image-level label, the JSON Line looks similar to the following.

```
{"source-ref": "s3://custom-labels-console-us-east-1-nnnnnnnnnn/gt-job/manifest/IMG_1133.png", "TestCLConsoleBucket": 0, "TestCLConsoleBucket-metadata": {"confidence": 0.95, "job-name": "labeling-job/testclconsolebucket", "class-name": "Echo Dot", "human-annotated": "yes", "creation-date": "2020-04-15T20:17:23.433061", "type": "groundtruth/image-classification"}}
```

The `source-ref` field indicates the location of the image. The JSON line also includes the image-level labels assigned to the image. For more information, see [the section called "Image-Level labels in manifest files" \(p. 249\)](#).

To assign image-level labels to an image

1. Get the get JSON Line for the existing image by using the `ListDatasetEntries`. For the `source-ref` field, specify the location of the image that you want to assign the label to. For more information, see [Listing dataset entries \(SDK\) \(p. 227\)](#).
2. Update the JSON Line returned in the previous step using the information at [Image-Level labels in manifest files \(p. 249\)](#).
3. Call `UpdateDatasetEntries` to update the image. For more information, see [Adding more images to a dataset \(p. 218\)](#).

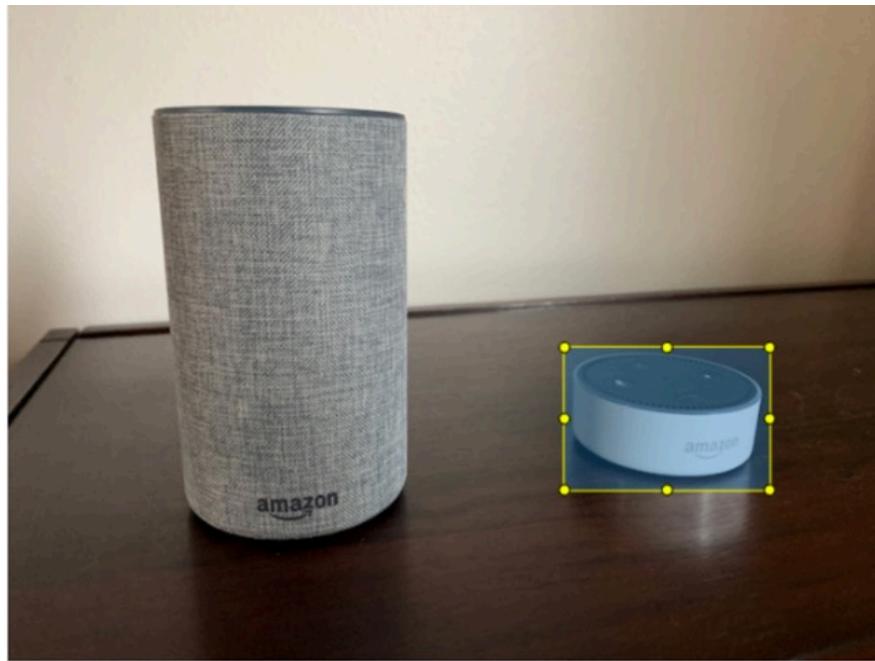
Labeling objects with bounding boxes

If you want your model to detect the location of objects within an image, you must identify what the object is and where it is in the image. A bounding box is a box that isolates an object in an image. You use bounding boxes to train a model to detect different objects in the same image. You identify the object by assigning a label to the bounding box.

Note

If you're training a model to find objects, scenes, and concepts with image-level labels, you don't need to do this step.

For example, if you want to train a model that detects Amazon Echo Dot devices, you draw a bounding box around each Echo Dot in an image and assign a label named *Echo Dot* to the bounding box. The following image shows a bounding box around an Echo Dot device. The image also contains an Amazon Echo without a bounding box.

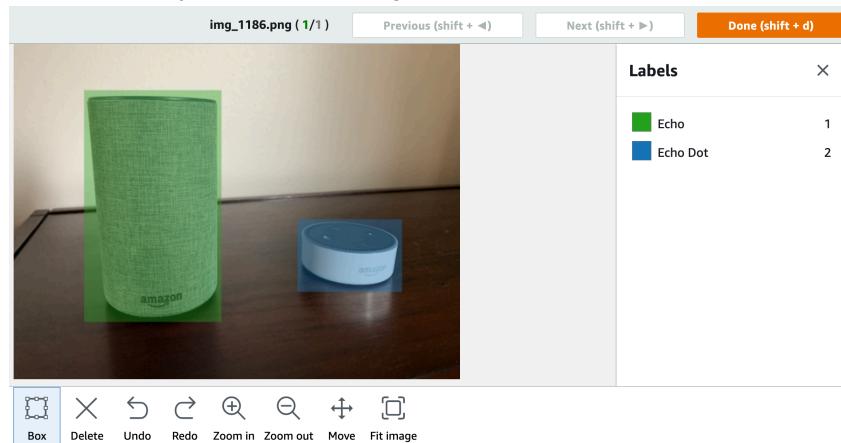


Locate objects with bounding boxes (Console)

In this procedure, you use the console to draw bounding boxes around the objects in your images. You also can identify objects within the image by assigning labels to the bounding box.

Before you can add bounding boxes, you must add at least one label to the dataset. For more information, see [Add new labels \(Console\) \(p. 100\)](#).

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. Choose **Use Custom Labels**.
3. Choose **Get started**.
4. In the left navigation pane, choose **Projects**.
5. In the **Projects** page, choose the project that you want to use. The details page for your project is displayed.
6. On the project details page, choose **Label images**
7. If you want to add bounding boxes to your training dataset images, choose the **Training** tab. Otherwise choose the **Test** tab to add bounding boxes to the test dataset images.
8. Choose **Start labeling** to enter labeling mode.
9. In the image gallery, choose the images that you want to add bounding boxes to.
10. Choose **Draw bounding box**. A series of tips are shown before the bounding box editor is displayed.
11. In the **Labels** pane on the right, select the label that you want to assign to a bounding box.
12. In the drawing tool, place your pointer at the top-left area of the desired object.
13. Press the left mouse button and draw a box around the object. Try to draw the bounding box as close as possible to the object.
14. Release the mouse button. The bounding box is highlighted.
15. Choose **Next** if you have more images to label. Otherwise, choose **Done** to finish labeling.



16. Repeat steps 1–7 until you have created a bounding box in each image that contains objects.
17. Choose **Save changes** to save your changes.
18. Choose **Exit** to exit labeling mode.

Locate objects with bounding boxes (SDK)

You can use the `UpdateDatasetEntries` API to add or update object location information for an image. `UpdateDatasetEntries` takes one or more JSON lines. Each JSON Line represents a single image. For object localization, a JSON Line looks similar to the following.

```
{"source-ref": "s3://bucket/images/IMG_1186.png", "bounding-box": {"image_size": [{"width": 640, "height": 480, "depth": 3}], "annotations": [{"class_id": 1, "top": 251, "left": 399, "width": 155, "height": 101}, {"class_id": 0, "top": 65, "left": 86, "width": 220, "height": 334}], "bounding-box-metadata": {"objects": [{"confidence": 1}, {"confidence": 1}], "class-map": {"0": "Echo", "1": "Echo Dot"}, "type": "groundtruth/object-detection", "human-annotated": "yes", "creation-date": "2013-11-18T02:53:27", "job-name": "my job"}}}
```

The `source-ref` field indicates the location of the image. The JSON line also includes labeled bounding boxes for each object on the image. For more information, see [the section called “Object localization in manifest files” \(p. 252\)](#).

To assign bounding boxes to an image

1. Get the get JSON Line for the existing image by using the `ListDatasetEntries`. For the `source-ref` field, specify the location of the image that you want to assign the image-level label to. For more information, see [Listing dataset entries \(SDK\) \(p. 227\)](#).
2. Update the JSON Line returned in the previous step using the information at [Object localization in manifest files \(p. 252\)](#).
3. Call `UpdateDatasetEntries` to update the image. For more information, see [Adding more images to a dataset \(p. 218\)](#).

Training an Amazon Rekognition Custom Labels model

You can train a model by using the Amazon Rekognition Custom Labels console, or by the Amazon Rekognition Custom Labels API. If model training fails, use the information in [Debugging a failed model training \(p. 116\)](#) to find the cause of the failure.

Note

You are charged for the amount of time that it takes to successfully train a model. Typically training takes from 30 minutes to 24 hours to complete. For more information, see [Training hours](#).

A new version of a model is created every time the model is trained. Amazon Rekognition Custom Labels creates a name for the model that is a combination of the project name and the timestamp for when the model is created.

To train your model, Amazon Rekognition Custom Labels makes a copy of your source training and test images. By default the copied images are encrypted at rest with a key that AWS owns and manages. You can also choose to use your own AWS KMS key. If you use your own KMS key, you need the following permissions on the KMS key.

- `kms:CreateGrant`
- `kms:DescribeKey`

For more information, see [AWS Key Management Service concepts](#). Your source images are unaffected.

You can use KMS server-side encryption (SSE-KMS) to encrypt the training and test images in your Amazon S3 bucket, before they are copied by Amazon Rekognition Custom Labels. To allow Amazon Rekognition Custom Labels access to your images, your account needs the following permissions on the KMS key.

- `kms:GenerateDataKey`
- `kms:Decrypt`

For more information, see [Protecting Data Using Server-Side Encryption with KMS keys Stored in AWS Key Management Service \(SSE-KMS\)](#).

Optionally, you can manage your models by using tags. For more information, see [Tagging a model \(p. 274\)](#).

After training a model, you can evaluate its performance and make improvements. For more information, see [Improving a trained Amazon Rekognition Custom Labels model \(p. 152\)](#).

Topics

- [Training a model \(Console\) \(p. 107\)](#)
- [Training a model \(SDK\) \(p. 109\)](#)

Training a model (Console)

You can use the Amazon Rekognition Custom Labels console to train a model.

Training requires a project with a training dataset and a test dataset. If your project doesn't have a test dataset, the Amazon Rekognition Custom Labels console splits the training dataset during training to create one for your project. The images chosen are a representative sampling and aren't used in the training dataset. We recommend splitting your training dataset only if you don't have an alternative test dataset that you can use. Splitting a training dataset reduces the number of images available for training.

Note

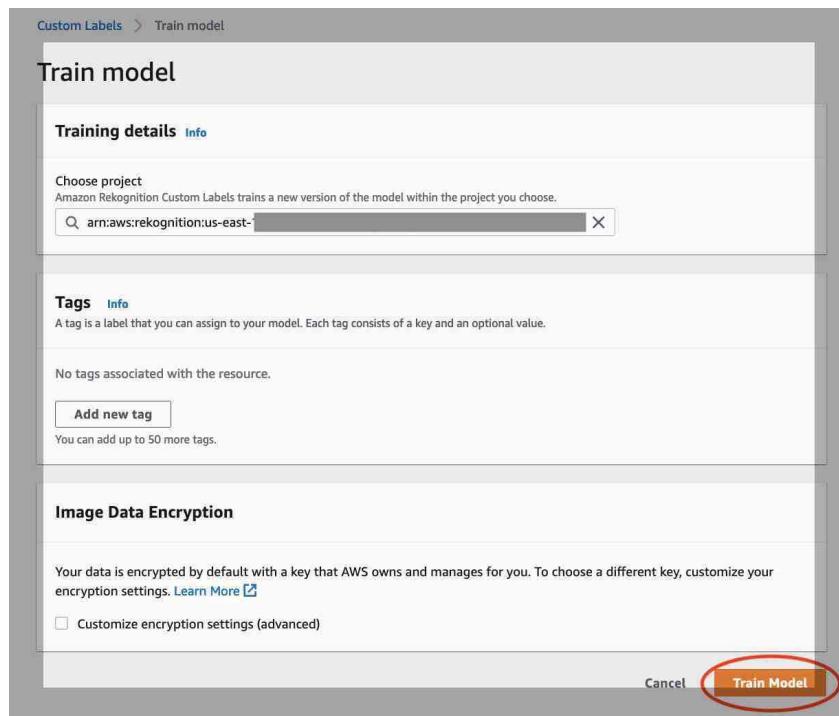
You are charged for the amount of time that it takes to train a model. For more information, see [Training hours](#).

To train your model (console)

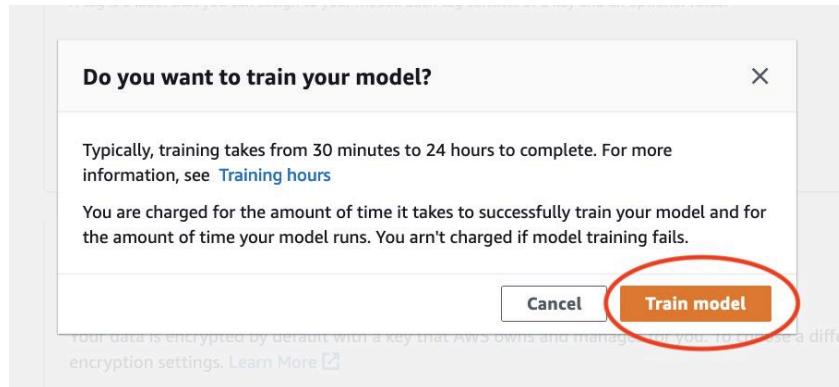
1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. Choose **Use Custom Labels**.
3. In the left navigation pane, choose **Projects**.
4. In the **Projects** page, choose the project that contains the model that you want to train.
5. On the **Project** page, choose **Train model**.



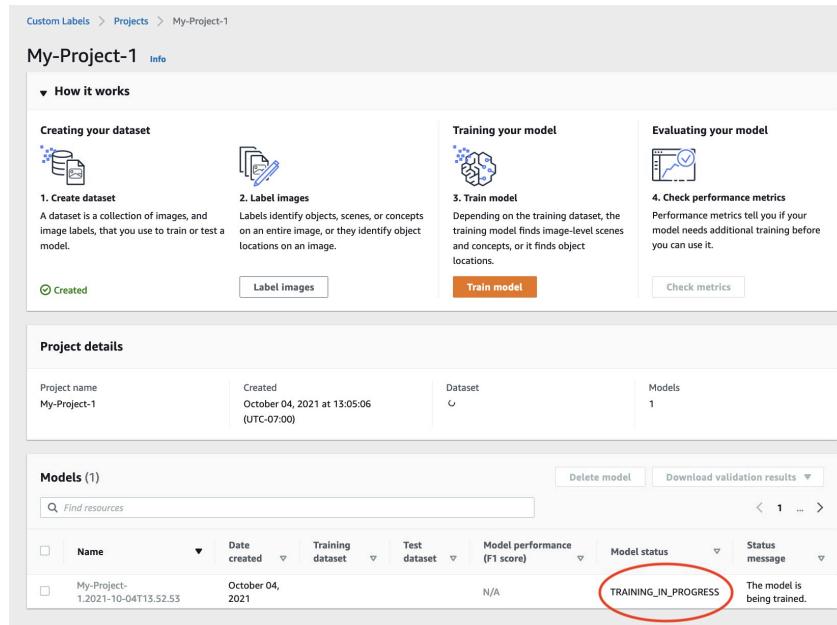
6. (Optional) If you want to use your own AWS KMS encryption key, do the following:
 - a. In **Image data encryption** choose **Customize encryption settings (advanced)**.
 - b. In **encryption.aws_kms_key** enter the Amazon Resource Name (ARN) of your key, or choose an existing AWS KMS key. To create a new key, choose **Create an AWS IMS key**.
7. (Optional) if you want to add tags to your model do the following:
 - a. In the **Tags** section, choose **Add new tag**.
 - b. Enter the following:
 - i. The name of the key in **Key**.
 - ii. The value of the key in **Value**.
 - c. To add more tags, repeat steps 6a and 6b.
 - d. (Optional) If you want to remove a tag, choose **Remove** next to the tag that you want to remove. If you are removing a previously saved tag, it is removed when you save your changes.
8. On the **Train model** page, Choose **Train model**. The Amazon Resource Name (ARN) for your project should be in the **Choose project** edit box. If not, enter the ARN for your project.



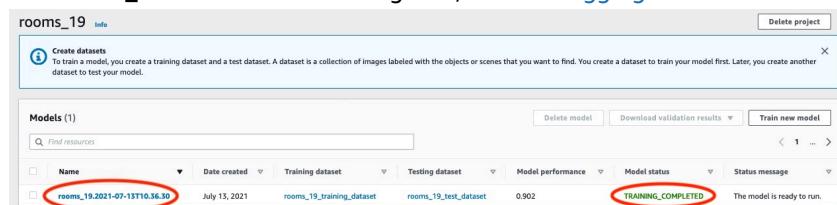
9. In the **Do you want to train your model?** dialog box, choose **Train model**.



10. In the **Models** section of the project page, you can check the current status in the **Model Status** column, where the training's in progress. Training a model takes a while to complete.



11. After training completes, choose the model name. Training is finished when the model status is **TRAINING_COMPLETED**. If training fails, read [Debugging a failed model training \(p. 116\)](#).



12. Next step: Evaluate your model. For more information, [Improving a trained Amazon Rekognition Custom Labels model \(p. 152\)](#).

Training a model (SDK)

You train a model by calling [CreateProjectVersion](#). To train a model, the following information is needed:

- Name – A unique name for the model version.
- Project ARN – The Amazon Resource Name (ARN) of the project that manages the model.
- Training results location – The Amazon S3 location where the results are placed. You can use the same location as the console Amazon S3 bucket, or you can choose a different location. We recommend choosing a different location because this allows you to set permissions and avoid potential naming conflicts with training output from using the Amazon Rekognition Custom Labels console.

Training uses the training and test datasets associated with project. For more information, see [Managing datasets \(p. 211\)](#).

Note

Optionally, you can specify training and test dataset manifest files that are external to a project. If you open the console after training a model with external manifest files, Amazon Rekognition Custom Labels creates the datasets for you by using the last set of manifest files used for training. You can no longer train a model version for the project by specifying external manifest files. For more information, see [CreateProjectVersion](#).

The response from `CreateProjectVersion` is an ARN that you use to identify the model version in subsequent requests. You can also use the ARN to secure the model version. For more information, see [Securing Amazon Rekognition Custom Labels projects \(p. 319\)](#).

Training a model version takes a while to complete. The Python and Java examples in this topic use waiters to wait for training to complete. A waiter is a utility method that polls for a particular state to occur. Alternatively, you can get the current status of training by calling `DescribeProjectVersions`. Training is completed when the `Status` field value is `TRAINING_COMPLETED`. After training is completed, you can evaluate model's quality by reviewing the evaluation results.

Topics

- [Training a model \(SDK\) \(p. 110\)](#)

Training a model (SDK)

The following example shows how to train a model by using the training and test datasets associated with a project.

To train a model (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with `AmazonRekognitionFullAccess` and permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following example code to train a project.

AWS CLI

The following example creates a model. The training dataset is split to create the testing dataset. Replace the following:

- `my_project_arn` with the Amazon Resource Name (ARN) of the project.
- `version_name` with a unique version name of your choosing.
- `output_bucket` with the name of the Amazon S3 bucket where Amazon Rekognition Custom Labels saves the training results.
- `output_folder` with the name of the folder where the training results are saved.
- (optional parameter) `--kms-key-id` with identifier for your AWS Key Management Service customer master key.

```
aws rekognition create-project-version\  
  --project-arn "project_arn"\\  
  --version-name "version_name"\\  
  --output-config '{"S3Bucket":"'output_bucket",  
  "S3KeyPrefix":"'output_folder"}'
```

Python

The following example creates a model. Supply the following command line arguments:

- `project_arn` – The Amazon Resource Name (ARN) of the project.
- `version_name` – A unique version name for the model of your choosing.

- `output_bucket` – the name of the Amazon S3 bucket where Amazon Rekognition Custom Labels saves the training results.
- `output_folder` – the name of the folder where the training results are saved.

Optionally, supply the following command line parameters to attach a tag to your model:

- `tag` – a tag name of your choosing that you want to attach to the model.
- `tag_value` the tag value.

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

from os import stat_result
import boto3
import argparse
import logging
import time
import json
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def train_model(rek_client, project_arn, version_name, output_bucket,
                output_folder, tag_key, tag_key_value):
    """
        Trains an Amazon Rekognition Custom Labels model.
        :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
        :param project_arn: The ARN of the project in which you want to train a model.
        :param version_name: A version for the model.
        :param output_bucket: The S3 bucket that hosts training output.
        :param output_folder: The path for the training output within output_bucket
        :param tag_key: The name of a tag to attach to the model. Pass None to exclude
        :param tag_key_value: The value of the tag. Pass None to exclude
    """

    try:
        #Train the model

        status=""
        logger.info(f"training model version {version_name} for project {project_arn}")

        output_config = json.loads(
            '{"S3Bucket": "'
            + output_bucket
            + '", "S3KeyPrefix": "'
            + output_folder
            + '" }'
        )

        tags=[]

        if tag_key!=None and tag_key_value !=None:
            tags = json.loads(
                '{"' + tag_key + '":"' + tag_key_value + '"}'
            )
    
```

```
        response=rek_client.create_project_version(
            ProjectArn=project_arn,
            VersionName=version_name,
            OutputConfig=output_config,
            Tags=tags
        )

        logger.info(f"Started training: {response['ProjectVersionArn']}")

        # Wait for the project version training to complete

        project_version_training_completed_waiter =
rek_client.get_waiter('project_version_training_completed')
        project_version_training_completed_waiter.wait(ProjectArn=project_arn,
VersionNames=[version_name])

        #Get the completion status

        describe_response=rek_client.describe_project_versions(ProjectArn=project_arn,
            VersionNames=[version_name])
        for model in describe_response['ProjectVersionDescriptions']:
            logger.info("Status: " + model['Status'])
            logger.info("Message: " + model['StatusMessage'])
            status=model['Status']

        logger.info(f"finished training")

        return response['ProjectVersionArn'], status

    except ClientError as err:
        logger.exception(f"Couldn't create model: {err.response['Error']}"
['Message']}")
        raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project in which you want to train a
model"
    )

    parser.add_argument(
        "version_name", help="A version name of your choosing."
    )

    parser.add_argument(
        "output_bucket", help="The S3 bucket that receives the training results."
    )

    parser.add_argument(
        "output_folder", help="The folder in the S3 bucket where training results
are stored."
    )

    parser.add_argument(
        "--tag_name", help="The name of a tag to attach to the model",
        required=False
    )

    parser.add_argument(
```

```
        "--tag_value", help="The value for the tag.", required=False
    )

def main():

    logging.basicConfig(level=logging.INFO, format"%(levelname)s: %(message)s")

    try:

        #get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Training model version {args.version_name} for project
{args.project_arn}")

        #Train the model
        rek_client=boto3.client('rekognition')

        model_arn, status=train_model(rek_client,
            args.project_arn,
            args.version_name,
            args.output_bucket,
            args.output_folder,
            args.tag_name,
            args.tag_value)

        print(f"Finished training model: {model_arn}")
        print(f"Status: {status}")

    except ClientError as err:
        logger.exception(f"Problem training model: {err}")
        print(f"Problem training model: {err}")
    except Exception as err:
        logger.exception(f"Problem training model: {err}")
        print(f"Problem training model: {err}")

if __name__ == "__main__":
    main()
```

Java

The following example trains a model. Supply the following command line arguments:

- **project_arn** – The Amazon Resource Name (ARN) of the project.
- **version_name** – A unique version name for the model of your choosing.
- **output_bucket** – the name of the Amazon S3 bucket where Amazon Rekognition Custom Labels saves the training results.
- **output_folder** – the name of the folder where the training results are saved.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)
```

```
import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
software.amazon.awssdk.services.rekognition.model.CreateProjectVersionRequest;
import
software.amazon.awssdk.services.rekognition.model.CreateProjectVersionResponse;
import
software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsRequest;
import
software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsResponse;
import software.amazon.awssdk.services.rekognition.model.OutputConfig;
import software.amazon.awssdk.services.rekognition.model.ProjectVersionDescription;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.waiters.RekognitionWaiter;

import java.net.URI;
import java.util.Optional;
import java.util.logging.Level;
import java.util.logging.Logger;

public class TrainModel {

    public static final Logger logger =
Logger.getLogger(TrainModel.class.getName());

    public static String trainMyModel(RekognitionClient rekClient, String
projectArn, String versionName,
        String outputBucket, String outputFolder) {

        try {

            OutputConfig outputConfig =
OutputConfig.builder().s3Bucket(outputBucket).s3KeyPrefix(outputFolder).build();

            logger.log(Level.INFO, "Training Model for project {0}", projectArn);
            CreateProjectVersionRequest createProjectVersionRequest =
CreateProjectVersionRequest.builder()
.projectArn(projectArn).versionName(versionName).outputConfig(outputConfig).build();

            CreateProjectVersionResponse response =
rekClient.createProjectVersion(createProjectVersionRequest);

            logger.log(Level.INFO, "Model ARN: {0}", response.projectVersionArn());
            logger.log(Level.INFO, "Training model...");

            // wait until training completes

            DescribeProjectVersionsRequest describeProjectVersionsRequest =
DescribeProjectVersionsRequest.builder()
.versionNames(versionName)
.projectArn(projectArn)
.build();

            RekognitionWaiter waiter = rekClient.waiter();

            WaiterResponse<DescribeProjectVersionsResponse> waiterResponse = waiter
.waitUntilProjectVersionTrainingCompleted(describeProjectVersionsRequest);

            Optional<DescribeProjectVersionsResponse> optionalResponse =
waiterResponse.matched().response();

            DescribeProjectVersionsResponse describeProjectVersionsResponse =
optionalResponse.get();
        }
    }
}
```

```
        for (ProjectVersionDescription projectVersionDescription :  
describeProjectVersionsResponse  
                .projectVersionDescriptions()) {  
            System.out.println("ARN: " +  
projectVersionDescription.projectVersionArn());  
            System.out.println("Status: " +  
projectVersionDescription.statusAsString());  
            System.out.println("Message: " +  
projectVersionDescription.statusMessage());  
        }  
  
        return response.projectVersionArn();  
    } catch (RekognitionException e) {  
        logger.log(Level.SEVERE, "Could not train model: {0}", e.getMessage());  
        throw e;  
    }  
}  
  
public static void main(String args[]) {  
  
    String versionName = null;  
    String projectArn = null;  
    String projectVersionArn = null;  
    String bucket = null;  
    String location = null;  
  
    final String USAGE = "\n" + "Usage: " + "<project_name> <version_name>  
<output_bucket> <output_folder>\n\n" + "Where:\n"  
        + "    project_arn - The ARN of the project that you want to use. \n  
\n"  
        + "    version_name - A version name for the model.\n\n"  
        + "    output_bucket - The S3 bucket in which to place the training  
output. \n\n"  
        + "    output_folder - The folder within the bucket that the  
training output is stored in. \n\n";  
  
    if (args.length != 4) {  
        System.out.println(USAGE);  
        System.exit(1);  
    }  
  
    projectArn = args[0];  
    versionName = args[1];  
    bucket = args[2];  
    location = args[3];  
  
    try {  
  
        // Get the Rekognition client  
        RekognitionClient rekClient = RekognitionClient.builder().build();  
  
        // Train model  
        projectVersionArn = trainMyModel(rekClient, projectArn, versionName,  
bucket, location);  
  
        System.out.println(String.format("Created model: %s for Project ARN:  
%s", projectVersionArn, projectArn));  
  
        rekClient.close();  
  
    } catch (RekognitionException rekError) {  
        logger.log(Level.SEVERE, "Rekognition client error: {0}",  
rekError.getMessage());  
    }  
}
```

```
        System.exit(1);
    }
}
```

3. If training fails, read [Debugging a failed model training \(p. 116\)](#).

Debugging a failed model training

You might encounter errors during model training. Amazon Rekognition Custom Labels reports training errors in the console and in the response from [DescribeProjectVersions](#).

Errors are either terminal (training can't continue), or they are non-terminal (training can continue). For errors that relate to the contents of the training and testing datasets, you can download the validation results (a [manifest summary \(p. 118\)](#) and [training and testing validation manifests \(p. 121\)](#)). Use the error codes in the validation results to find further information in this section. This section also provides information for manifest file errors (terminal errors that happen before the manifest file contents are validated).

Note

A manifest is the file used to store the contents of a dataset.

You can fix some errors by using the Amazon Rekognition Custom Labels console. Other errors might require you to make updates to the training or testing manifest files. You might need to make other changes, such as IAM permissions. For more information, see the documentation for individual errors.

Terminal errors

Terminal errors stop the training of a model. There are 3 categories of terminal training errors – service errors, manifest file errors, and manifest content errors.

In the console, Amazon Rekognition Custom Labels shows terminal errors for a model in the **Status message** column of the projects page.

Projects (853) Info					
Delete Download validation results Train new model Create project					
Name	Versions	Date created	Model performance	Model status	Status message
rt...	1	2020-10-05	0.608	TRAINING_COMPLETED	The model is ready to run.
test_1	19	2020-09-29		STOPPED	The model has stopped running.
test_4		2020-09-30	0.261	STOPPED	The model has stopped running.
test_20		2020-10-05	N/A	TRAINING_FAILED	Amazon Rekognition experienced a service issue.

If you using the AWS SDK, you can find out if a terminal manifest file error or a terminal manifest content error has occurred by checking the response from [DescribeProjectVersions](#). In this case, the Status value is TRAINING_FAILED and StatusMessage field contains the error.

Service errors

Terminal service errors occur when Amazon Rekognition experiences a service issue and can't continue training. For example, the failure of another service that Amazon Rekognition Custom Labels depends upon. Amazon Rekognition Custom Labels reports service errors in the console as *Amazon Rekognition experienced a service issue*. If you use the AWS SDK, service errors that occur during training are raised as an `InternalServerError` exception by [CreateProjectVersion](#) and [DescribeProjectVersions](#).

If a service error occurs, retry training of the model. If training continues to fail, contact [AWS Support](#) and include any error information reported with the service error.

Terminal manifest file errors

Manifest file errors are terminal errors, in the training and testing datasets, that happen at the file level, or across multiple files. Manifest file errors are detected before the contents of the training and testing datasets are validated. Manifest file errors prevent the reporting of [non-terminal validation errors \(p. 118\)](#). For example, an empty training manifest file generates an *The manifest file is empty* error. Since the file is empty, no non-terminal JSON Line validation errors can be reported. The manifest summary is also not created.

You must fix manifest file errors before you can train your model.

The following lists the manifest file errors.

- [The manifest file extension or contents are invalid. \(p. 127\)](#)
- [The manifest file is empty. \(p. 128\)](#)
- [The manifest file size exceeds the maximum supported size. \(p. 128\)](#)
- [Unable to write to output S3 bucket. \(p. 128\)](#)
- [The S3 bucket permissions are incorrect. \(p. 128\)](#)

Terminal manifest content errors

Manifest content errors are terminal errors that relate to the content within a manifest. For example, if you get the error [The manifest file contains insufficient labeled images per label to perform auto-split \(p. 130\)](#), training can't finish as there aren't enough labeled images in the training dataset to create a testing dataset.

As well as being reported in the console and in the response from `DescribeProjectVersions`, the error is reported in the manifest summary along with any other terminal manifest content errors. For more information, see [Understanding the manifest summary \(p. 118\)](#).

Non terminal JSON Line errors are also reported in separate training and testing validation results manifests. The non-terminal JSON Line errors found by Amazon Rekognition Custom Labels are not necessarily related to the manifest content error(s) that stop training. For more information, see [Understanding training and testing validation result manifests \(p. 121\)](#).

You must fix manifest content errors before you can train your model.

The following are the error messages for manifest content errors.

- [The manifest file contains too many invalid rows. \(p. 129\)](#)
- [The manifest file contains images from multiple S3 buckets. \(p. 129\)](#)
- [Invalid owner id for images S3 bucket. \(p. 130\)](#)
- [The manifest file contains insufficient labeled images per label to perform auto-split. \(p. 130\)](#)
- [The manifest file has too few labels. \(p. 131\)](#)
- [The manifest file has too many labels. \(p. 132\)](#)
- [Less than {}% label overlap between the training and testing manifest files. \(p. 133\)](#)
- [The manifest file has too few usable labels. \(p. 133\)](#)
- [Less than {}% usable label overlap between the training and testing manifest files. \(p. 134\)](#)
- [Failed to copy images from S3 bucket. \(p. 135\)](#)

Non terminal JSON line validation errors

JSON Line validation errors are non-terminal errors that don't require Amazon Rekognition Custom Labels to stop training a model.

JSON Line validation errors are not shown in the console.

In the training and testing datasets, a JSON Line represents the training or testing information for a single image. Validation errors in a JSON Line, such as an invalid image, are reported in the training and testing validation manifests. Amazon Rekognition Custom Labels completes training using the other, valid, JSON Lines that are in the manifest. For more information, see [Understanding training and testing validation result manifests \(p. 121\)](#). For information about validation rules, see [Validation rules for manifest files \(p. 255\)](#).

Note

Training fails if there are too many JSON Line errors.

We recommend that you also fix non-terminal JSON Line errors errors as they can potentially cause future errors or impact your model training.

Amazon Rekognition Custom Labels can generate the following non-terminal JSON Line validation errors.

- [The source-ref key is missing. \(p. 136\)](#)
- [The format of the source-ref value is invalid. \(p. 136\)](#)
- [No label attributes found. \(p. 136\)](#)
- [The format of the label attribute `{}` is invalid. \(p. 137\)](#)
- [The format of the label attributemetadata is invalid. \(p. 138\)](#)
- [No valid label attributes found. \(p. 139\)](#)
- [One or more bounding boxes has a missing confidence value. \(p. 139\)](#)
- [One of more class ids is missing from the class map. \(p. 140\)](#)
- [The JSON Line has an invalid format. \(p. 141\)](#)
- [The image is invalid. Check S3 path and/or image properties. \(p. 141\)](#)
- [The bounding box has off frame values. \(p. 143\)](#)
- [The height and width of the bounding box is too small. \(p. 145\)](#)
- [There are more bounding boxes than the allowed maximum. \(p. 145\)](#)
- [No valid annotations found. \(p. 144\)](#)

Understanding the manifest summary

The manifest summary contains the following information.

- Error information about [Terminal manifest content errors \(p. 117\)](#) encountered during validation.
- Error location information for [Non terminal JSON line validation errors \(p. 118\)](#) in the training and testing datasets.
- Error statistics such as the total number of invalid JSON Lines found in the training and testing datasets.

The manifest summary is created during training if there are no [Terminal manifest file errors \(p. 117\)](#). To get the location of the manifest summary file (`manifest_summary.json`), see [Getting the validation results \(p. 124\)](#).

Note

[Service errors \(p. 116\)](#) and [manifest file errors \(p. 117\)](#) are not reported in the manifest summary. For more information, see [Terminal errors \(p. 116\)](#).

For information about specific manifest content errors, see [Terminal manifest content errors \(p. 129\)](#).

Manifest summary file format

A manifest file has 2 sections, **statistics** and **errors**.

statistics

statistics contains information about the errors in the training and testing datasets.

- **training** – statistics and errors found in the training dataset.
- **testing** – statistics and errors found in the testing dataset.

Objects in the **errors** array contain the error code and message for manifest content errors.

The **error_line_indices** array contains the line numbers for each JSON Line in the training or test manifest that has an error. For more information, see [Fixing training errors \(p. 126\)](#).

errors

Errors spanning both the training and testing dataset. For example, an [ERROR_INSUFFICIENT_USABLE_LABEL_OVERLAP \(p. 134\)](#) occurs when there is isn't enough usable labels that overlap the training and testing datasets.

```
{  
  "statistics": {  
    "training": {  
      "use_case": String, # Possible values are IMAGE_LEVEL_LABELS,  
      OBJECT_LOCALIZATION and NOT_DETERMINED  
      "total_json_lines": Number, # Total number json lines (images) in the  
      training manifest.  
      "valid_json_lines": Number, # Total number of JSON Lines (images) that  
      can be used for training.  
      "invalid_json_lines": Number, # Total number of invalid JSON Lines. They  
      are not used for training.  
      "ignored_json_lines": Number, # JSON Lines that have a valid schema but  
      have no annotations. The aren't used for training and aren't counted as invalid.  
      "error_json_line_indices": List[int], # Contains a list of line numbers for  
      JSON line errors in the training dataset.  
      "errors": [  
        {  
          "code": String, # Error code for a training manifest content error.  
          "message": String # Description for a training manifest content  
          error.  
        }  
      ]  
    },  
    "testing": {  
      "use_case": String, # Possible values are IMAGE_LEVEL_LABELS,  
      OBJECT_LOCALIZATION and NOT_DETERMINED  
      "total_json_lines": Number, # Total number json lines (images) in the  
      manifest.  
      "valid_json_lines": Number, # Total number of JSON Lines (images) that can  
      be used for testing.  
    }  
  }  
}
```

```
        "invalid_json_lines": Number, # Total number of invalid JSON Lines. They
        are not used for testing.
        "ignored_json_lines": Number, # JSON Lines that have a valid schema but
        have no annotations. They aren't used for testing and aren't counted as invalid.
        "error_json_line_indices": List[int], # contains a list of error record
        line numbers in testing dataset.
        "errors": [
            {
                "code": String, # # Error code for a testing manifest content
                error.
                "message": String # Description for a testing manifest content
                error.
            }
        ],
        "errors": [
            {
                "code": String, # # Error code for errors that span the training and testing
                datasets.
                "message": String # Description of the error.
            }
        ]
    }
```

Example manifest summary

The following example is a partial manifest summary that shows a terminal manifest content error ([ERROR_TOO_MANY_INVALID_ROWS_IN_MANIFEST \(p. 129\)](#)). The `error_json_line_indices` array contains the line numbers of non-terminal JSON Line errors in the corresponding training or testing validation manifest.

```
{
    "errors": [],
    "statistics": {
        "training": {
            "use_case": "NOT_DETERMINED",
            "total_json_lines": 301,
            "valid_json_lines": 146,
            "invalid_json_lines": 155,
            "ignored_json_lines": 0,
            "errors": [
                {
                    "code": "ERROR_TOO_MANY_INVALID_ROWS_IN_MANIFEST",
                    "message": "The manifest file contains too many invalid rows."
                }
            ],
            "error_json_line_indices": [
                15,
                16,
                17,
                22,
                23,
                24,
                .
                .
                .
                300
            ]
        },
        "testing": {
            "use_case": "NOT_DETERMINED",
        }
    }
}
```

```
        "total_json_lines": 15,  
        "valid_json_lines": 13,  
        "invalid_json_lines": 2,  
        "ignored_json_lines": 0,  
        "errors": [],  
        "error_json_line_indices": [  
            13,  
            15  
        ]  
    }  
}  
}
```

Understanding training and testing validation result manifests

During training, Amazon Rekognition Custom Labels creates validation result manifests to hold non-terminal JSON Line errors. The validation results manifests are copies of the training and testing datasets with error information added. You can access the validation manifests after training completes. For more information, see [Getting the validation results \(p. 124\)](#). Amazon Rekognition Custom Labels also creates a manifest summary that includes overview information for JSON Line errors, such as error locations and JSON Line error counts. For more information, see [Understanding the manifest summary \(p. 118\)](#).

Note

Validation results (Training and Testing Validation Result Manifests and Manifest Summary) are only created if there are no [Terminal manifest file errors \(p. 117\)](#).

A manifest contains JSON Lines for each image in the dataset. Within the validation results manifests, JSON Line error information is added to the JSON Lines where errors occur.

A JSON Line error is a non-terminal error related to a single image. A non-terminal validation error can invalidate the entire JSON Line or just a portion. For example, if the image referenced in a JSON Line is not in PNG or JPG format, an [ERROR_INVALID_IMAGE \(p. 141\)](#) error occurs and the entire JSON Line is excluded from training. Training continues with other valid JSON Lines.

Within a JSON Line, an error might mean the JSON Line can still be used for training. For example, if the left value for one of four bounding boxes associated with a label is negative, the model is still trained using the other valid bounding boxes. JSON Line error information is returned for the invalid bounding box ([ERROR_INVALID_BOUNDING_BOX \(p. 143\)](#)). In this example, the error information is added to the annotation object where the error occurs.

Warning errors, such as [WARNING_NO_ANNOTATIONS \(p. 147\)](#), aren't used for training and count as ignored JSON lines (`ignored_json_lines`) in the manifest summary. For more information, see [Understanding the manifest summary \(p. 118\)](#). Additionally, ignored JSON Lines don't count towards the 20% error threshold for training and testing.

For information about specific non-terminal data validation errors, see [Non-Terminal JSON Line Validation Errors \(p. 135\)](#).

Note

If there are too many data validation errors, training is stopped and a [ERROR_TOO_MANY_INVALID_ROWS_IN_MANIFEST \(p. 129\)](#) terminal error is reported in the manifest summary.

For information about correcting JSON Line errors, see [Fixing training errors \(p. 126\)](#).

JSON line error format

Amazon Rekognition Custom Labels adds non-terminal validation error information to image level and object localization format JSON Lines. For more information, see [the section called “Creating a manifest file” \(p. 242\)](#).

Image Level Errors

The following example shows the `Error` arrays in an image level JSON Line. There are two sets of errors. Errors related to label attribute metadata (in this example, `sport-metadata`) and errors related to the image. An error includes an error code (`code`), error message (`message`). For more information, see [Image-Level labels in manifest files \(p. 249\)](#).

```
{  
  "source-ref": String,  
  "sport": Number,  
  "sport-metadata": {  
    "class-name": String,  
    "confidence": Float,  
    "type": String,  
    "job-name": String,  
    "human-annotated": String,  
    "creation-date": String,  
    "errors": [  
      {  
        "code": String, # error codes for label  
        "message": String # Description and additional contextual details of the  
error  
      }  
    ]  
  },  
  "errors": [  
    {  
      "code": String, # error codes for image  
      "message": String # Description and additional contextual details of the error  
    }  
  ]  
}
```

Object localization errors

The following example shows the error arrays in an object localization JSON Line. The JSON Line contains an `Errors` array information for fields in the following JSON Line sections. Each `Error` object includes the error code and the error message.

- *label attribute* – Errors for the label attribute fields. See bounding-box in the example.
 - *annotations* – Annotation errors (bounding boxes) are stored in the annotations array inside the label attribute.
 - *label attribute-metadata* – Errors for the label attribute metadata. See bounding-box-metadata in the example.
 - *image* – Errors not related to the label attribute, annotation, and label attribute metadata fields.

For more information, see [Object localization in manifest files \(p. 252\)](#).

```
{  
  "source-ref": String,  
  "bounding-box": {  
    "image size": [
```

```
{  
    "width": Int,  
    "height": Int,  
    "depth": Int,  
}  
]  
,  
"annotations": [  
    {  
        "class_id": Int,  
        "left": Int,  
        "top": Int,  
        "width": Int,  
        "height": Int,  
        "errors": [ # annotation field errors  
            {  
                "code": String, # annotation field error code  
                "message": String # Description and additional contextual details  
of the error  
            }  
        ]  
    }  
],  
"errors": [ #label attribute field errors  
    {  
        "code": String, # error code  
        "message": String # Description and additional contextual details of the  
error  
    }  
]  
},  
"bounding-box-metadata": {  
    "objects": [  
        {  
            "confidence": Float  
        }  
    ],  
    "class-map": {  
        String: String  
    },  
    "type": String,  
    "human-annotated": String,  
    "creation-date": String,  
    "job-name": String,  
    "errors": [ #metadata field errors  
        {  
            "code": String, # error code  
            "message": String # Description and additional contextual details of the  
error  
        }  
    ]  
},  
"errors": [ # image errors  
    {  
        "code": String, # error code  
        "message": String # Description and additional contextual details of the error  
    }  
]  
}  
}
```

Example JSON line error

The following object localization JSON Line (formatted for readability) shows an [ERROR_BOUNDING_BOX_TOO_SMALL \(p. 145\)](#) error. In this example, the bounding box dimensions (height and width) aren't greater than 1 x 1.

```
{  
  "source-ref": "s3://bucket/Manifests/images/199940-1791.jpg",  
  "bounding-box": [  
    "image_size": [  
      {  
        "width": 3000,  
        "height": 3000,  
        "depth": 3  
      }  
    ],  
    "annotations": [  
      {  
        "class_id": 1,  
        "top": 0,  
        "left": 0,  
        "width": 1,  
        "height": 1,  
        "errors": [  
          {  
            "code": "ERROR_BOUNDING_BOX_TOO_SMALL",  
            "message": "The height and width of the bounding box is too small."  
          }  
        ]  
      },  
      {  
        "class_id": 0,  
        "top": 65,  
        "left": 86,  
        "width": 220,  
        "height": 334  
      }  
    ]  
  ],  
  "bounding-box-metadata": {  
    "objects": [  
      {  
        "confidence": 1  
      },  
      {  
        "confidence": 1  
      }  
    ],  
    "class-map": {  
      "0": "Echo",  
      "1": "Echo Dot"  
    },  
    "type": "groundtruth/object-detection",  
    "human-annotated": "yes",  
    "creation-date": "2019-11-20T02:57:28.288286",  
    "job-name": "my job"  
  }  
}
```

Getting the validation results

The validation results contain error information for [Terminal manifest content errors \(p. 117\)](#) and [Non terminal JSON line validation errors \(p. 118\)](#). There are three validation results files.

- *training_manifest_with_validation.json* – A copy of the training dataset manifest file with JSON Line error information added.
- *testing_manifest_with_validation.json* – A copy of the testing dataset manifest file with JSON Line error information added.

- *manifest_summary.json* – A summary of manifest content errors and JSON Line errors found in the training and testing datasets. For more information, see [Understanding the manifest summary \(p. 118\)](#).

For information about the contents of the training and testing validation manifests, see [Debugging a failed model training \(p. 116\)](#).

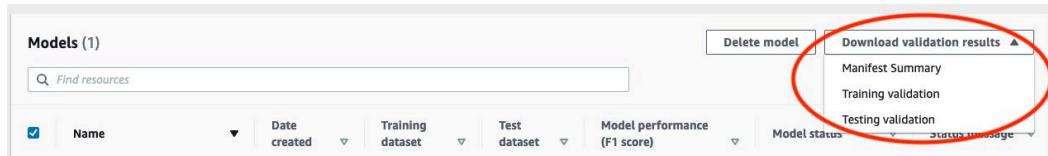
Note

- The validation results are created only if no [Terminal manifest file errors \(p. 117\)](#) are generated during training.
- If a [service error \(p. 116\)](#) occurs after the training and testing manifest are validated, the validation results are created, but the response from [DescribeProjectVersions](#) doesn't include the validation results file locations.

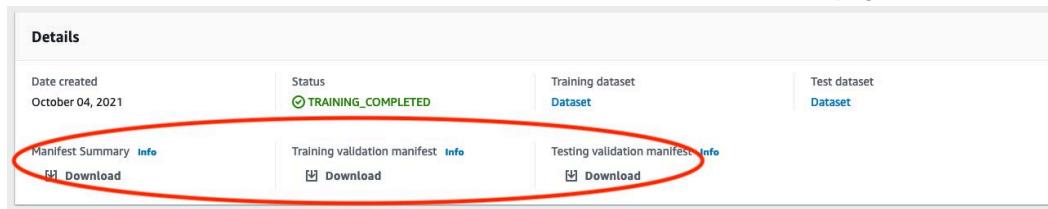
After training completes or fails, you can download the validation results by using the Amazon Rekognition Custom Labels console or get the Amazon S3 bucket location by calling [DescribeProjectVersions](#) API.

Getting validation results (Console)

If you are using the console to train your model, you can download the validation results from a project's list of models, as shown in the following diagram.



You can also access download the validation results from a model's details page.



For more information, see [Training a model \(Console\) \(p. 107\)](#).

Getting validation results (SDK)

After model training completes, Amazon Rekognition Custom Labels stores the validation results in the Amazon S3 bucket specified during training. You can get the S3 bucket location by calling the [DescribeProjectVersions](#) API, after training completes. To train a model, see [Training a model \(SDK\) \(p. 109\)](#).

A `ValidationData` object is returned for the training dataset (`TrainingDataResult`) and the testing dataset (`TestingDataResult`). The manifest summary is returned in `ManifestSummary`.

After you get the Amazon S3 bucket location, you can download the validation results. For more information, see [How do I download an object from an S3 bucket?](#). You can also use the `GetObject` operation.

To get validation data (SDK)

1. If you haven't already:

- a. Create or update an IAM user with `AmazonRekognitionFullAccess` and permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
- b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following example to get the location of the validation results.

Python

Replace `project_arn` with the Amazon Resource Name (ARN) of the project that contains the model. For more information, see [Managing an Amazon Rekognition Custom Labels project \(p. 200\)](#). Replace `version_name` with the name of the model version. For more information, see [Training a model \(SDK\) \(p. 109\)](#).

```
import boto3
import io
from io import BytesIO
import sys
import json

def describe_model(project_arn, version_name):
    client=boto3.client('rekognition')

    response=client.describe_project_versions(ProjectArn=project_arn,
                                                VersionNames=[version_name])

    for model in response['ProjectVersionDescriptions']:
        print(json.dumps(model,indent=4,default=str))

def main():

    project_arn='project_arn'
    version_name='version_name'

    describe_model(project_arn, version_name)

if __name__ == "__main__":
    main()
```

3. In the program output, note the `Validation` field within the `TestingDataResult` and `TrainingDataResult` objects. The manifest summary is in `ManifestSummary`.

Fixing training errors

You use the manifest summary to identify [Terminal manifest content errors \(p. 117\)](#) and [Non terminal JSON line validation errors \(p. 118\)](#) encountered during training. You must fix manifest content errors. We recommend that you also fix non-terminal JSON Line errors. For information about specific errors, see [Non-Terminal JSON Line Validation Errors \(p. 135\)](#) and [Terminal manifest content errors \(p. 129\)](#).

You can make fixes to the training or testing dataset used for training. Alternatively, you can make the fixes in the training and testing validation manifest files and use them to train the model.

After you make your fixes, you need to import the updated manifest(s) and retrain the model. For more information, see [Creating a manifest file \(p. 242\)](#).

The following procedure shows you how to use the manifest summary to fix terminal manifest content errors. The procedure also shows you how to locate and fix JSON Line errors in the training and testing validation manifests.

To fix Amazon Rekognition Custom Labels training errors

1. Download the validation results files. The file names are `training_manifest_with_validation.json`, `testing_manifest_with_validation.json` and `manifest_summary.json`. For more information, see [Getting the validation results \(p. 124\)](#).
2. Open the manifest summary file (`manifest_summary.json`).
3. Fix any errors in the manifest summary. For more information, see [Understanding the manifest summary \(p. 118\)](#).
4. In the manifest summary, iterate through the `error_line_indices` array in `training` and fix the errors in `training_manifest_with_validation.json` at the corresponding JSON Line numbers. For more information, see [the section called "Understanding training and testing validation result manifests" \(p. 121\)](#).
5. Iterate through the `error_line_indices` array in `testing` and fix the errors in `testing_manifest_with_validation.json` at the corresponding JSON Line numbers.
6. Retrain the model using the validation manifest files as the training and testing datasets. For more information, see [the section called "Training a model" \(p. 106\)](#).

If you are using the AWS SDK and choose to fix the errors in the training or the test validation data manifest files, use the location of the validation data manifest files in the `TrainingData` and `TestingData` input parameters to `CreateProjectVersion`. For more information, see [Training a model \(SDK\) \(p. 109\)](#).

JSON line error precedence

The following JSON Line errors are detected first. If any of these errors occur, validation of JSON Line errors is stopped. You must fix these errors before you can fix any of the other JSON Line errors

- `MISSING_SOURCE_REF`
- `ERROR_INVALID_SOURCE_REF_FORMAT`
- `ERROR_NO_LABEL_ATTRIBUTES`
- `ERROR_INVALID_LABEL_ATTRIBUTE_FORMAT`
- `ERROR_INVALID_LABEL_ATTRIBUTE_METADATA_FORMAT`
- `ERROR_MISSING_BOUNDING_BOX_CONFIDENCE`
- `ERROR_MISSING_CLASS_MAP_ID`
- `ERROR_INVALID_JSON_LINE`

Terminal manifest file errors

This topic describes the [Terminal manifest file errors \(p. 117\)](#). Manifest file errors do not have an associated error code. The validation results manifests are not created when a terminal manifest file error occurs. For more information, see [Understanding the manifest summary \(p. 118\)](#). Terminal manifest errors prevent the reporting of [Non-Terminal JSON Line Validation Errors \(p. 135\)](#).

The manifest file extension or contents are invalid.

The training or testing manifest file doesn't have a file extension or its contents are invalid.

To fix error *The manifest file extension or contents are invalid.*

- Check the following possible causes in both the training and testing manifest files.
 - The manifest file is missing a file extension. By convention the file extension is `.manifest`.

- The Amazon S3 bucket or key for the manifest file couldn't be found.

The manifest file is empty.

The training or testing manifest file used for training exists, but it is empty. The manifest file needs a JSON Line for each image that you use for training and testing.

To fix error *The manifest file is empty.*

1. Check which of the training or testing manifests are empty.
2. Add JSON Lines to the empty manifest file. For more information, see [Creating a manifest file \(p. 242\)](#). Alternatively, create a new dataset with the console. For more information, see the section called "Creating datasets (Console)" (p. 59).

The manifest file size exceeds the maximum supported size.

The training or testing manifest file size (in bytes) is too large. For more information, see [Guidelines and quotas in Amazon Rekognition Custom Labels \(p. 323\)](#). A manifest file can have less than the maximum number of JSON Lines and still exceed the maximum file size.

You can't use the Amazon Rekognition Custom Labels console to fix error *The manifest file size exceeds the maximum supported size.*

To fix error *The manifest file size exceeds the maximum supported size.*

1. Check which of the training and testing manifests exceed the maximum file size.
2. Reduce the number of JSON Lines in the manifest files that are too large. For more information, see [Creating a manifest file \(p. 242\)](#).

The S3 bucket permissions are incorrect.

Amazon Rekognition Custom Labels doesn't have permissions to one or more of the buckets containing the training and testing manifest files.

You can't use the Amazon Rekognition Custom Labels console to fix this error.

To fix error *The S3 bucket permissions are incorrect.*

- Check the permissions for the bucket(s) containing the training and testing manifests. For more information, see [Step 4: Set up Amazon Rekognition Custom Labels permissions \(p. 6\)](#).

Unable to write to output S3 bucket.

The service is unable to generate the training output files.

To fix error *Unable to write to output S3 bucket.*

- Check that the Amazon S3 bucket information in the [OutputConfig](#) input parameter to [CreateProjectVersion](#) is correct.

You can't use the Amazon Rekognition Custom Labels console to fix this error.

Terminal manifest content errors

This topic describes the [Terminal manifest content errors \(p. 117\)](#) reported in the manifest summary. The manifest summary includes an error code and message for each detected error. For more information, see [Understanding the manifest summary \(p. 118\)](#). Terminal manifest content errors don't stop the reporting of [Non terminal JSON line validation errors \(p. 118\)](#).

ERROR_TOO_MANY_INVALID_ROWS_IN_MANIFEST

Error message

The manifest file contains too many invalid rows.

More information

An `ERROR_TOO_MANY_INVALID_ROWS_IN_MANIFEST` error occurs if there are too many JSON Lines that contain invalid content.

You can't use the Amazon Rekognition Custom Labels console to fix an `ERROR_TOO_MANY_INVALID_ROWS_IN_MANIFEST` error.

To fix `ERROR_TOO_MANY_INVALID_ROWS_IN_MANIFEST`

1. Check the manifest for JSON Line errors. For more information, see [Understanding training and testing validation result manifests \(p. 121\)](#).
2. Fix JSON Lines that have errors For more information, see [Non-Terminal JSON Line Validation Errors \(p. 135\)](#).

ERROR_IMAGES_IN_MULTIPLE_S3_BUCKETS

Error message

The manifest file contains images from multiple S3 buckets.

More information

A manifest can only reference images stored in a single bucket. Each JSON Line stores the Amazon S3 location of an image location in the value of `source-ref`. In the following example, the bucket name is `my-bucket`.

```
"source-ref": "s3://my-bucket/images/sunrise.png"
```

You can't use the Amazon Rekognition Custom Labels console to fix this error.

To fix `ERROR_IMAGES_IN_MULTIPLE_S3_BUCKETS`

- Ensure that all your images are in the same Amazon S3 bucket and that the value of `source-ref` in every JSON Line references the bucket where your images are stored. Alternatively, choose a preferred Amazon S3 bucket and remove the JSON Lines where `source-ref` doesn't reference your preferred bucket.

ERROR_INVALID_PERMISSIONS_IMAGES_S3_BUCKET

Error message

The permissions for the images S3 bucket are invalid.

More information

The permissions on the Amazon S3 bucket that contains the images are incorrect.

You can't use the Amazon Rekognition Custom Labels console to fix this error.

To fix ERROR_INVALID_PERMISSIONS_IMAGES_S3_BUCKET

- Check the permissions of the bucket containing the images. The value of the `source-ref` for an image contains the bucket location.

ERROR_INVALID_IMAGES_S3_BUCKET_OWNER

Error message

Invalid owner id for images S3 bucket.

More information

The owner of the bucket that contains the training or test images is different from the owner of the bucket that contains the training or test manifest. You can use the following command to find the owner of a bucket.

```
aws s3api get-bucket-acl --bucket bucket name
```

The OWNER ID must match for the buckets that store the images and manifest files.

To fix ERROR_INVALID_IMAGES_S3_BUCKET_OWNER

- Choose the desired owner of the training, testing, output, and image buckets. The owner must have permissions to use Amazon Rekognition Custom Labels.
- For each bucket not currently owned by the desired owner, create a new Amazon S3 bucket owned by the preferred owner.
- Copy the old bucket contents to the new bucket. For more information, see [How can I copy objects between Amazon S3 buckets?](#).

You can't use the Amazon Rekognition Custom Labels console to fix this error.

ERROR_INSUFFICIENT_IMAGES_PER_LABEL_FOR_AUTOSPLIT

Error message

The manifest file contains insufficient labeled images per label to perform auto-split.

More information

During model training, you can create a testing dataset by using 20% of the images from the training dataset. ERROR_INSUFFICIENT_IMAGES_PER_LABEL_FOR_AUTOSPLIT occurs when there aren't enough images to create an acceptable testing dataset.

You can't use the Amazon Rekognition Custom Labels console to fix this error.

To fix **ERROR_INSUFFICIENT_IMAGES_PER_LABEL_FOR_AUTOSPLIT**

- Add more labeled image to your training dataset. You can add images in the Amazon Rekognition Custom Labels console by adding images to the training dataset, or by adding JSON Lines to your training manifest. For more information, see [Managing datasets \(p. 211\)](#).

ERROR_MANIFEST_TOO_FEW_LABELS

Error message

The manifest file has too few labels.

More information

Training and testing datasets have a required minimum number of labels. The minimum depends on if the dataset trains/tests a model to detect image-level labels (classification) or if the model detects object locations. If the training dataset is split to create a testing dataset, the number of labels in the dataset is determined after the training dataset is split. For more information, see [Guidelines and quotas in Amazon Rekognition Custom Labels \(p. 323\)](#).

To fix **ERROR_MANIFEST_TOO_FEW_LABELS (console)**

1. Add more new labels to the dataset. For more information, see [Managing labels \(p. 100\)](#).
2. Add the new labels to images in the dataset. If your model detects image-level labels, see [Assigning image-level labels to an image \(p. 102\)](#). If your model detects object locations, see the section called "Labeling objects with bounding boxes" (p. 104).

To fix **ERROR_MANIFEST_TOO_FEW_LABELS (JSON Line)**

- Add JSON Lines for new images that have new labels. For more information, see [Creating a manifest file \(p. 242\)](#). If your model detects image-level labels, you add new labels names to the class-name field. For example, the label for the following image is *Sunrise*.

```
{  
  "source-ref": "s3://bucket/images/sunrise.png",  
  "testdataset-classification_Sunrise": 1,  
  "testdataset-classification_Sunrise-metadata": {  
    "confidence": 1,  
    "job-name": "labeling-job/testdataset-classification_Sunrise",  
    "class-name": "Sunrise",  
    "human-annotated": "yes",  
    "creation-date": "2018-10-18T22:18:13.527256",  
    "type": "groundtruth/image-classification"  
  }  
}
```

If your model detects object locations, add new labels to the class-map, as shown in the following example.

```
{  
  "source-ref": "s3://custom-labels-bucket/images/IMG_1186.png",  
  "bounding-box": {
```

```
  "image_size": [{"  
    "width": 640,  
    "height": 480,  
    "depth": 3  
}],  
  "annotations": [{"  
    "class_id": 1,  
    "top": 251,  
    "left": 399,  
    "width": 155,  
    "height": 101  
}, {  
    "class_id": 0,  
    "top": 65,  
    "left": 86,  
    "width": 220,  
    "height": 334  
}]  
},  
  "bounding-box-metadata": {  
    "objects": [{"  
        "confidence": 1  
}, {  
        "confidence": 1  
}],  
    "class-map": {  
        "0": "Echo",  
        "1": "Echo Dot"  
    },  
    "type": "groundtruth/object-detection",  
    "human-annotated": "yes",  
    "creation-date": "2018-10-18T22:18:13.527256",  
    "job-name": "my job"  
}  
}
```

You need to map the class map table to the bounding box annotations. For more information, see [Object localization in manifest files \(p. 252\)](#).

ERROR_MANIFEST_TOO_MANY_LABELS

Error message

The manifest file has too many labels.

More information

The number of unique labels in the manifest (dataset) is more than the allowed limit. If the training dataset is split to create a testing dataset, the number of labels is determined after the split.

To fix ERROR_MANIFEST_TOO_MANY_LABELS (Console)

- Remove labels from the dataset. For more information, see [Managing labels \(p. 100\)](#). The labels are automatically removed from the images and bounding boxes in your dataset.

To fix ERROR_MANIFEST_TOO_MANY_LABELS (JSON Line)

- Manifests with image level JSON Lines – If the image has a single label, remove the JSON Lines for images that use the desired label. If the JSON Line contains multiple labels, remove only the JSON

object for the desired label. For more information, see [Adding multiple image-level labels to an image \(p. 251\)](#).

Manifests with object location JSON Lines – Remove the bounding box and associated label information for the label that you want to remove. Do this for each JSON Line that contains the desired label. You need to remove the label from the `class-map` array and corresponding objects in the `objects` and `annotations` array. For more information, see [Object localization in manifest files \(p. 252\)](#).

ERROR_INSUFFICIENT_LABEL_OVERLAP

Error message

Less than {}% label overlap between the training and testing manifest files.

More information

There is less than 50% overlap between the testing dataset label names and the training dataset label names.

To fix ERROR_INSUFFICIENT_LABEL_OVERLAP (Console)

- Remove labels from the training dataset. Alternatively, add more common labels to your testing dataset. For more information, see [Managing labels \(p. 100\)](#). The labels are automatically removed from the images and bounding boxes in your dataset.

To fix ERROR_INSUFFICIENT_LABEL_OVERLAP by removing labels from the training dataset (JSON Line)

- Manifests with image level JSON Lines – If the image has a single label, remove the JSON Line for the image that use the desired label. If the JSON Line contains multiple labels, remove only the JSON object for the desired label. For more information, see [Adding multiple image-level labels to an image \(p. 251\)](#). Do this for each JSON Line in the manifest that contains the label that you want to remove.

Manifests with object location JSON Lines – Remove the bounding box and associated label information for the label that you want to remove. Do this for each JSON Line that contains the desired label. You need to remove the label from the `class-map` array and corresponding objects in the `objects` and `annotations` array. For more information, see [Object localization in manifest files \(p. 252\)](#).

To fix ERROR_INSUFFICIENT_LABEL_OVERLAP by adding common labels to the testing dataset (JSON Line)

- Add JSON Lines to the testing dataset that include images labeled with labels already in the training dataset. For more information, see [Creating a manifest file \(p. 242\)](#).

ERROR_MANIFEST_TOO_FEW_USABLE_LABELS

Error message

The manifest file has too few usable labels.

More information

A training manifest can contain JSON Lines in image-level label format and in object location format. Depending on type of JSON Lines found in the training manifest, Amazon Rekognition Custom Labels chooses to create a model that detects image-level labels, or a model that detects object locations. Amazon Rekognition Custom Labels filters out valid JSON records for JSON Lines that are not in the chosen format. **ERROR_MANIFEST_TOO_FEW_USABLE_LABELS** occurs when the number of labels in the chosen model type manifest is insufficient to train the model.

A minimum of 1 label is required to train a model that detects image-level labels. A minimum of 2 labels is required to train a model that object locations.

To fix **ERROR_MANIFEST_TOO_FEW_USABLE_LABELS** (Console)

1. Check the `use_case` field in the manifest summary.
2. Add more labels to the training dataset for the use case (image level or object localization) that matches the value of `use_case`. For more information, see [Managing labels \(p. 100\)](#). The labels are automatically removed from the images and bounding boxes in your dataset.

To fix **ERROR_MANIFEST_TOO_FEW_USABLE_LABELS** (JSON Line)

1. Check the `use_case` field in the manifest summary.
2. Add more labels to the training dataset for the use case (image level or object localization) that matches the value of `use_case`. For more information, see [Creating a manifest file \(p. 242\)](#).

ERROR_INSUFFICIENT_USABLE_LABEL_OVERLAP

Error message

Less than {}% usable label overlap between the training and testing manifest files.

More information

A training manifest can contain JSON Lines in image-level label format and in object location format. Depending on the formats found in the training manifest, Amazon Rekognition Custom Labels chooses to create a model that detects image-level labels, or a model that detects object locations. Amazon Rekognition Custom Labels doesn't use valid JSON records for JSON Lines that are not in the chosen model format. **ERROR_INSUFFICIENT_USABLE_LABEL_OVERLAP** occurs when there is less than 50% overlap between the testing and training labels that are used.

To fix **ERROR_INSUFFICIENT_USABLE_LABEL_OVERLAP** (Console)

- Remove labels from the training dataset. Alternatively, add more common labels to your testing dataset. For more information, see [Managing labels \(p. 100\)](#). The labels are automatically removed from the images and bounding boxes in your dataset.

To fix **ERROR_INSUFFICIENT_USABLE_LABEL_OVERLAP** by removing labels from the training dataset (JSON Line)

- Datasets used to detect image-level labels – If the image has a single label, remove the JSON Line for the image that use the desired label. If the JSON Line contains multiple labels, remove only the JSON object for the desired label. For more information, see [Adding multiple image-level labels to an image \(p. 251\)](#). Do this for each JSON Line in the manifest that contains the label that you want to remove.

Datasets used to detect object locations – Remove the bounding box and associated label information for the label that you want to remove. Do this for each JSON Line that contains the desired label. You need to remove the label from the `class-map` array and corresponding objects in the `objects` and `annotations` array. For more information, see [Object localization in manifest files \(p. 252\)](#).

To fix `ERROR_INSUFFICIENT_USABLE_LABEL_OVERLAP` by adding common labels to the testing dataset (JSON Line)

- Add JSON Lines to the testing dataset that include images labeled with labels already in the training dataset. For more information, see [Creating a manifest file \(p. 242\)](#).

ERROR_FAILED_IMAGES_S3_COPY

Error message

Failed to copy images from S3 bucket.

More information

The service wasn't able to copy any of the images in your dataset.

You can't use the Amazon Rekognition Custom Labels console to fix this error.

To fix `ERROR_FAILED_IMAGES_S3_COPY`

1. Check the permissions of your images.
2. If you are using AWS KMS, check the bucket policy. For more information, see [Decrypting files encrypted with AWS Key Management Service \(p. 9\)](#).

The manifest file has too many terminal errors.

There are too many JSON lines with terminal content errors.

To fix `ERROR_TOO_MANY_RECORDS_IN_ERROR`

- Reduce the number of JSON Lines (images) with terminal content errors. For more information, see [Terminal manifest content errors \(p. 129\)](#).

You can't use the Amazon Rekognition Custom Labels console to fix this error.

Non-Terminal JSON Line Validation Errors

This topic lists the non-terminal JSON Line validation errors reported by Amazon Rekognition Custom Labels during training. The errors are reported in the training and testing validation manifest. For more information, see [Understanding training and testing validation result manifests \(p. 121\)](#). You can fix a non-terminal JSON Line error by updating the JSON Line in the training or test manifest file. You can also remove the JSON Line from the manifest, but doing so might reduce the quality of your model. If there are many non-terminal validation errors, you might find it easier to recreate the manifest file. Validation errors typically occur in manually created manifest files. For more information, see [Creating a manifest file \(p. 242\)](#). For information about fixing validation errors, see [Fixing training errors \(p. 126\)](#). Some errors can be fixed by using the Amazon Rekognition Custom Labels console.

ERROR_MISSING_SOURCE_REF

Error message

The source-ref key is missing.

More information

The JSON Line source-ref field provides the Amazon S3 location of an image. This error occurs when the source-ref key is missing or is misspelt. This error typically occurs in manually created manifest files. For more information, see [Creating a manifest file \(p. 242\)](#).

To fix ERROR_MISSING_SOURCE_REF

1. Check that the source-ref key is present and is spelt correctly. A complete source-ref key and value is similar to the following. is "source-ref": "s3://bucket/path/image".
2. Update or the source-ref key in the JSON Line. Alternatively, remove, the JSON Line from the manifest file.

You can't use the Amazon Rekognition Custom Labels console to fix this error.

ERROR_INVALID_SOURCE_REF_FORMAT

Error message

The format of the source-ref value is invalid.

More information

The source-ref key is present in the JSON Line, but the schema of the Amazon S3 path is incorrect. For example, the path is https://..... instead of S3://..... An ERROR_INVALID_SOURCE_REF_FORMAT error typically occurs in manually created manifest files. For more information, see [Creating a manifest file \(p. 242\)](#).

To fix ERROR_INVALID_SOURCE_REF_FORMAT

1. Check that the schema is "source-ref": "s3://bucket/path/image". For example, "source-ref": "s3://custom-labels-console-us-east-1-1111111111/images/000000242287.jpg".
2. Update, or remove, the JSON Line in the manifest file.

You can't use the Amazon Rekognition Custom Labels console to fix this ERROR_INVALID_SOURCE_REF_FORMAT.

ERROR_NO_LABEL_ATTRIBUTES

Error message

No label attributes found.

More information

The label attribute or the label attribute -metadata key name (or both) is invalid or missing. In the following example, ERROR_NO_LABEL_ATTRIBUTES occurs whenever the bounding-box or bounding-box-metadata key (or both) is missing. For more information, see [Creating a manifest file \(p. 242\)](#).

```
{  
  "source-ref": "s3://custom-labels-bucket/images/IMG_1186.png",  
  "bounding-box": {  
    "image_size": [{  
      "width": 640,  
      "height": 480,  
      "depth": 3  
    }],  
    "annotations": [{  
      "class_id": 1,  
      "top": 251,  
      "left": 399,  
      "width": 155,  
      "height": 101  
    }, {  
      "class_id": 0,  
      "top": 65,  
      "left": 86,  
      "width": 220,  
      "height": 334  
    }]  
  },  
  "bounding-box-metadata": {  
    "objects": [{  
      "confidence": 1  
    }, {  
      "confidence": 1  
    }],  
    "class-map": {  
      "0": "Echo",  
      "1": "Echo Dot"  
    },  
    "type": "groundtruth/object-detection",  
    "human-annotated": "yes",  
    "creation-date": "2018-10-18T22:18:13.527256",  
    "job-name": "my job"  
  }  
}
```

A `ERROR_NO_LABEL_ATTRIBUTES` error typically occurs in a manually created manifest file. For more information, see [Creating a manifest file \(p. 242\)](#).

To fix `ERROR_NO_LABEL_ATTRIBUTES`

1. Check that label attribute identifier and label attribute identifier -metadata keys are present and that the key names are spelt correctly.
2. Update, or remove, the JSON Line in the manifest file.

You can't use the Amazon Rekognition Custom Labels console to fix `ERROR_NO_LABEL_ATTRIBUTES`.

[ERROR_INVALID_LABEL_ATTRIBUTE_FORMAT](#)

Error message

The format of the label attribute {} is invalid.

More information

The schema for the label attribute key is missing or invalid. An `ERROR_INVALID_LABEL_ATTRIBUTE_FORMAT` error typically occurs in manually created manifest files. For more information, see [Creating a manifest file \(p. 242\)](#).

To fix **ERROR_INVALID_LABEL_ATTRIBUTE_FORMAT**

1. Check that the JSON Line section for the label attribute key is correct. In the following example object location example, the `image_size` and `annotations` objects must be correct. The label attribute key is named `bounding-box`.

```
"bounding-box": {  
  "image_size": [{  
    "width": 640,  
    "height": 480,  
    "depth": 3  
  }],  
  "annotations": [{  
    "class_id": 1,  
    "top": 251,  
    "left": 399,  
    "width": 155,  
    "height": 101  
  }, {  
    "class_id": 0,  
    "top": 65,  
    "left": 86,  
    "width": 220,  
    "height": 334  
  }]  
},
```

2. Update, or remove, the JSON Line in the manifest file.

You can't use the Amazon Rekognition Custom Labels console to fix this error.

ERROR_INVALID_LABEL_ATTRIBUTE_METADATA_FORMAT

Error message

The format of the label attribute metadata is invalid.

More information

The schema for the label attribute metadata key is missing or invalid. An `ERROR_INVALID_LABEL_ATTRIBUTE_METADATA_FORMAT` error typically occurs in manually created manifest files. For more information, see [Creating a manifest file \(p. 242\)](#).

To fix **ERROR_INVALID_LABEL_ATTRIBUTE_FORMAT**

1. Check that the JSON Line schema for the label attribute metadata key is similar to the following example. The label attribute metadata key is named `bounding-box-metadata`.

```
"bounding-box-metadata": {  
  "objects": [{  
    "confidence": 1  
  }, {  
    "confidence": 1  
  }],  
  "class-map": {  
    "0": "Echo",  
    "1": "Echo Dot"  
  },  
  "type": "groundtruth/object-detection",
```

```
  "human-annotated": "yes",
  "creation-date": "2018-10-18T22:18:13.527256",
  "job-name": "my job"
}
```

2. Update, or remove, the JSON Line in the manifest file.

You can't use the Amazon Rekognition Custom Labels console to fix this error.

ERROR_NO_VALID_LABEL_ATTRIBUTES

Error message

No valid label attributes found.

More information

No valid label attributes were found in the JSON Line. Amazon Rekognition Custom Labels checks both the label attribute and the label attribute identifier. An `ERROR_INVALID_LABEL_ATTRIBUTE_FORMAT` error typically occurs in manually created manifest files. For more information, see [Creating a manifest file \(p. 242\)](#).

If a JSON Line isn't in a supported SageMaker manifest format, Amazon Rekognition Custom Labels marks the JSON Line as invalid and an `ERROR_NO_VALID_LABEL_ATTRIBUTES` error is reported. Currently, Amazon Rekognition Custom Labels supports classification job and bounding box formats. For more information, see [Creating a manifest file \(p. 242\)](#).

To fix `ERROR_NO_VALID_LABEL_ATTRIBUTES`

1. Check that the JSON for the label attribute key and label attribute metadata is correct.
2. Update, or remove, the JSON Line in the manifest file. For more information, see [the section called "Creating a manifest file" \(p. 242\)](#).

You can't use the Amazon Rekognition Custom Labels console to fix this error.

ERROR_MISSING_BOUNDING_BOX_CONFIDENCE

Error message

One or more bounding boxes has a missing confidence value.

More information

The confidence key is missing for one or more object location bounding boxes. The confidence key for a bounding box is in the label attribute metadata, as shown in the following example. A `ERROR_MISSING_BOUNDING_BOX_CONFIDENCE` error typically occurs in manually created manifest files. For more information, see [the section called "Object localization in manifest files" \(p. 252\)](#).

```
"bounding-box-metadata": {
  "objects": [
    {
      "confidence": 1
    },
    {
      "confidence": 1
    }
  ],
}
```

To fix **ERROR_MISSING_BOUNDING_BOX_CONFIDENCE**

1. Check that the objects array in the label attribute contains the same number of confidence keys as there are objects in the label attribute annotations array.
2. Update, or remove, the JSON Line in the manifest file.

You can't use the Amazon Rekognition Custom Labels console to fix this error.

ERROR_MISSING_CLASS_MAP_ID

Error message

One or more class ids is missing from the class map.

More information

The `class_id` in an annotation (bounding box) object doesn't have a matching entry in the label attribute metadata class map (`class-map`). For more information, see [Object localization in manifest files \(p. 252\)](#). A `ERROR_MISSING_CLASS_MAP_ID` error typically occurs in manually created manifest files.

To fix **ERROR_MISSING_CLASS_MAP_ID**

1. Check that the `class_id` value in each annotation (bounding box) object has a corresponding value in the `class-map` array, as shown in the following example. The `annotations` array and `class_map` array should have the same number of elements.

```
{  
  "source-ref": "s3://custom-labels-bucket/images/IMG_1186.png",  
  "bounding-box": {  
    "image_size": [{  
      "width": 640,  
      "height": 480,  
      "depth": 3  
    }],  
    "annotations": [{  
      "class_id": 1,  
      "top": 251,  
      "left": 399,  
      "width": 155,  
      "height": 101  
    }, {  
      "class_id": 0,  
      "top": 65,  
      "left": 86,  
      "width": 220,  
      "height": 334  
    }]  
  },  
  "bounding-box-metadata": {  
    "objects": [{  
      "confidence": 1  
    }, {  
      "confidence": 1  
    }],  
    "class-map": {  
      "0": "Echo",  
      "1": "Echo Dot"  
    },  
  }  
}
```

```
    "type": "groundtruth/object-detection",
    "human-annotated": "yes",
    "creation-date": "2018-10-18T22:18:13.527256",
    "job-name": "my job"
}
}
```

2. Update, or remove, the JSON Line in the manifest file.

You can't use the Amazon Rekognition Custom Labels console to fix this error.

ERROR_INVALID_JSON_LINE

Error message

The JSON Line has an invalid format.

More information

An unexpected character was found in the JSON Line. The JSON Line is replaced with a new JSON Line that contains only the error information. An ERROR_INVALID_JSON_LINE error typically occurs in manually created manifest files. For more information, see [the section called "Object localization in manifest files" \(p. 252\)](#).

You can't use the Amazon Rekognition Custom Labels console to fix this error.

To fix ERROR_INVALID_JSON_LINE

1. Open the manifest file and navigate to the JSON Line where the ERROR_INVALID_JSON_LINE error occurs.
2. Check that the JSON Line doesn't contain invalid characters and that required ; or , characters are not missing.
3. Update, or remove, the JSON Line in the manifest file.

ERROR_INVALID_IMAGE

Error message

The image is invalid. Check S3 path and/or image properties.

More information

The file referenced by source-ref is not a valid image. Potential causes include the image aspect ratio, the size of the image, and the image format.

For more information, see [Guidelines and quotas \(p. 323\)](#).

To fix ERROR_INVALID_IMAGE

1. Check the following.
 - The aspect ratio of the image is less than 20:1.
 - The size of the image is greater than 15 MB
 - The image is in PNG or JPEG format.
 - The path to the image in source-ref is correct.
 - The minimum image dimension of the image is greater 64 pixels x 64 pixels.

- The maximum image dimension of the image is less than 4096 pixels x 4096 pixels.
2. Update, or remove, the JSON Line in the manifest file.

You can't use the Amazon Rekognition Custom Labels console to fix this error.

ERROR_INVALID_IMAGE_DIMENSION

Error message

The image dimension(s) do not conform to allowed dimensions.

More information

The image referenced by `source-ref` doesn't conform to the allowed image dimensions. The minimum dimension is 64 pixels. The maximum dimension is 4096 pixels. `ERROR_INVALID_IMAGE_DIMENSION` is reported for images with bounding boxes.

For more information, see [Guidelines and quotas \(p. 323\)](#).

To fix `ERROR_INVALID_IMAGE_DIMENSION` (Console)

1. Update the image in the Amazon S3 bucket with dimensions that Amazon Rekognition Custom Labels can process.
2. In the Amazon Rekognition Custom Labels console, do the following:
 - a. Remove the existing bounding boxes from the image.
 - b. Re-add the bounding boxes to the image.
 - c. Save your changes.

For more information, [Labeling objects with bounding boxes \(p. 104\)](#).

To fix `ERROR_INVALID_IMAGE_DIMENSION` (SDK)

1. Update the image in the Amazon S3 bucket with dimensions that Amazon Rekognition Custom Labels can process.
2. Get the existing JSON Line for the image by calling [ListDatasetEntries](#). For the `SourceRefContains` input parameter specify the Amazon S3 location and filename of the image.
3. Call [UpdateDatasetEntries](#) and provide the JSON line for the image. Make sure the value of `source-ref` matches the image location in the Amazon S3 bucket. Update the bounding box annotations to match the bounding box dimensions needed for the updated image.

```
{  
  "source-ref": "s3://custom-labels-bucket/images/IMG_1186.png",  
  "bounding-box": {  
    "image_size": [{  
      "width": 640,  
      "height": 480,  
      "depth": 3  
    }],  
    "annotations": [{  
      "class_id": 1,  
      "top": 251,  
      "left": 399,  
      "width": 155,  
      "height": 101  
    }, {
```

```
        "class_id": 0,
        "top": 65,
        "left": 86,
        "width": 220,
        "height": 334
    }],
},
"bounding-box-metadata": {
    "objects": [
        {
            "confidence": 1
        },
        {
            "confidence": 1
        }
    ],
    "class-map": {
        "0": "Echo",
        "1": "Echo Dot"
    },
    "type": "groundtruth/object-detection",
    "human-annotated": "yes",
    "creation-date": "2013-11-18T02:53:27",
    "job-name": "my job"
}
}
```

ERROR_INVALID_BOUNDING_BOX

Error message

The bounding box has off frame values.

More information

The bounding box information specifies an image that is either off the image frame or contains negative values.

For more information, see [Guidelines and quotas \(p. 323\)](#).

To fix ERROR_INVALID_BOUNDING_BOX

1. Check the values of the bounding boxes in the annotations array.

```
"bounding-box": {
    "image_size": [
        "width": 640,
        "height": 480,
        "depth": 3
    ],
    "annotations": [
        {
            "class_id": 1,
            "top": 251,
            "left": 399,
            "width": 155,
            "height": 101
        }
    ],
}
```

2. Update, or alternatively remove, the JSON Line from the manifest file.

You can't use the Amazon Rekognition Custom Labels console to fix this error.

ERROR_NO_VALID_ANNOTATIONS

Error message

No valid annotations found.

More information

None of the annotation objects in the JSON Line contain valid bounding box information.

To fix ERROR_NO_VALID_ANNOTATIONS

1. Update the annotations array to include valid bounding box objects. Also, check that corresponding bounding box information (confidence and class_map) in the label attribute metadata is correct. For more information, see [Object localization in manifest files \(p. 252\)](#).

```
{  
  "source-ref": "s3://custom-labels-bucket/images/IMG_1186.png",  
  "bounding-box": {  
    "image_size": [{  
      "width": 640,  
      "height": 480,  
      "depth": 3  
    }],  
    "annotations": [  
      {  
        "class_id": 1,      #annotation object  
        "top": 251,  
        "left": 399,  
        "width": 155,  
        "height": 101  
      }, {  
        "class_id": 0,  
        "top": 65,  
        "left": 86,  
        "width": 220,  
        "height": 334  
      }]  
    },  
    "bounding-box-metadata": {  
      "objects": [  
        >{  
          "confidence": 1          #confidence object  
        },  
        {  
          "confidence": 1  
        }],  
      "class-map": {  
        "0": "Echo",    #label  
        "1": "Echo Dot"  
      },  
      "type": "groundtruth/object-detection",  
      "human-annotated": "yes",  
      "creation-date": "2018-10-18T22:18:13.527256",  
      "job-name": "my job"  
    }  
}
```

2. Update, or alternatively remove, the JSON Line from the manifest file.

You can't use the Amazon Rekognition Custom Labels console to fix this error.

ERROR_BOUNDING_BOX_TOO_SMALL

Error message

The height and width of the bounding box is too small.

More information

The bounding box dimensions (height and width) have to be greater than 1 x 1 pixels.

During training, Amazon Rekognition Custom Labels resizes an image if any of its dimensions are greater than 1280 pixels (the source images aren't affected). The resulting bounding box heights and widths must be greater than 1 x 1 pixels. A bounding box location is stored in the annotations array of an object location JSON Line. For more information, see [Object localization in manifest files \(p. 252\)](#)

```
"bounding-box": {  
  "image_size": [{  
    "width": 640,  
    "height": 480,  
    "depth": 3  
  }],  
  "annotations": [{  
    "class_id": 1,  
    "top": 251,  
    "left": 399,  
    "width": 155,  
    "height": 101  
  }]  
},
```

The error information is added to the annotation object.

To fix ERROR_BOUNDING_BOX_TOO_SMALL

- Choose one of the following options.
 - Increase the size of bounding boxes that are too small.
 - Remove bounding boxes that are too small. For information about removing a bounding box, see [ERROR_TOO_MANY_BOUNDING_BOXES \(p. 145\)](#).
 - Remove the image (JSON Line) from the manifest.

ERROR_TOO_MANY_BOUNDING_BOXES

Error message

There are more bounding boxes than the allowed maximum.

More information

There are more bounding boxes than the allowed limit (50). You can remove excess bounding boxes in the Amazon Rekognition Custom Labels console, or you can remove them from the JSON Line.

To fix ERROR_TOO_MANY_BOUNDING_BOXES (Console).

1. Decide which bounding boxes to remove.
2. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.

3. Choose **Use Custom Labels**.
4. Choose **Get started**.
5. In the left navigation pane, choose the project that contains the dataset that you want to use.
6. In the **Datasets** section, choose the dataset that you want to use.
7. In the dataset gallery page, choose **Start labeling** to enter labeling mode.
8. Choose the image that you want to remove bounding boxes from.
9. Choose **Draw bounding box**.
10. In the drawing tool, choose the bounding box that you want to delete.
11. Press the delete key on your keyboard to delete the bounding box.
12. Repeat the previous 2 steps until you have deleted enough bounding boxes.
13. Choose **Done**
14. Choose **Save changes** to save your changes.
15. Choose **Exit** to exit labeling mode.

To fix **ERROR_TOO_MANY_BOUNDING_BOXES (JSON Line)**.

1. Open the manifest file and navigate to the JSON Line where the **ERROR_TOO_MANY_BOUNDING_BOXES** error occurs.
2. Remove the following for each bounding box that you want to remove.
 - Remove the required annotation object from annotations array.
 - Remove the corresponding confidence object from the objects array in the label attribute metadata.
 - If no longer used by other bounding boxes, remove the label from the class-map.

Use the following example to identify which items to remove.

```
{  
  "source-ref": "s3://custom-labels-bucket/images/IMG_1186.png",  
  "bounding-box": {  
    "image_size": [{  
      "width": 640,  
      "height": 480,  
      "depth": 3  
    }],  
    "annotations": [  
      {  
        "class_id": 1,      #annotation object  
        "top": 251,  
        "left": 399,  
        "width": 155,  
        "height": 101  
      }, {  
        "class_id": 0,  
        "top": 65,  
        "left": 86,  
        "width": 220,  
        "height": 334  
      }]  
    },  
    "bounding-box-metadata": {  
      "objects": [  
        >{  
          "confidence": 1      #confidence object  
        },  
        >{  
          "confidence": 0.95  
        }  
      ]  
    }  
}
```

```
{  
  "confidence": 1  
},  
  "class-map": {  
    "0": "Echo",      #label  
    "1": "Echo Dot"  
  },  
  "type": "groundtruth/object-detection",  
  "human-annotated": "yes",  
  "creation-date": "2018-10-18T22:18:13.527256",  
  "job-name": "my job"  
}  
}
```

WARNING_UNANNOTATED_RECORD

Warning Message

Record is unannotated.

More information

An image added to a dataset by using the Amazon Rekognition Custom Labels console wasn't labeled. The JSON line for the image isn't used for training.

```
{  
  "source-ref": "s3://bucket/images/IMG_1186.png",  
  "warnings": [  
    {  
      "code": "WARNING_UNANNOTATED_RECORD",  
      "message": "Record is unannotated."  
    }  
  ]  
}
```

To fix WARNING_UNANNOTATED_RECORD

- Label the image by using the Amazon Rekognition Custom Labels console. For instructions, see [Assigning image-level labels to an image \(p. 102\)](#).

WARNING_NO_ANNOTATIONS

Warning Message

No annotations provided.

More information

A JSON Line in Object Localization format doesn't contain any bounding box information, despite being annotated by a human (human-annotated = yes). The JSON Line is valid, but isn't used for training. For more information, see [Understanding training and testing validation result manifests \(p. 121\)](#).

```
{  
  "source-ref": "s3://bucket/images/IMG_1186.png",  
  "bounding-box": {
```

```
  "image_size": [
    {
      "width": 640,
      "height": 480,
      "depth": 3
    }
  ],
  "annotations": [
  ],
  "warnings": [
    {
      "code": "WARNING_NO_ATTRIBUTE_ANNOTATIONS",
      "message": "No attribute annotations were found."
    }
  ]
},
"bounding-box-metadata": {
  "objects": [
  ],
  "class-map": {
  },
  "type": "groundtruth/object-detection",
  "human-annotated": "yes",
  "creation-date": "2013-11-18 02:53:27",
  "job-name": "my job"
},
"warnings": [
  {
    "code": "WARNING_NO_ANNOTATIONS",
    "message": "No annotations were found."
  }
]
}
```

To fix **WARNING_NO_ANNOTATIONS**

- Choose one of the following options.
 - Add the bounding box (annotations) information to the JSON Line. For more information, see [Object localization in manifest files \(p. 252\)](#).
 - Remove the image (JSON Line) from the manifest.

WARNING_NO_ATTRIBUTE_ANNOTATIONS

Warning Message

No attribute annotations provided.

More information

A JSON Line in Object Localization format doesn't contain any bounding box annotation information, despite being annotated by a human (human-annotated = yes). The annotations array is not present or is not populated. The JSON Line is valid, but isn't used for training. For more information, see [Understanding training and testing validation result manifests \(p. 121\)](#).

```
{
```

```
"source-ref": "s3://bucket/images/IMG_1186.png",
"bounding-box": [
    "image_size": [
        {
            "width": 640,
            "height": 480,
            "depth": 3
        }
    ],
    "annotations": [
        ],
    "warnings": [
        {
            "code": "WARNING_NO_ATTRIBUTE_ANNOTATIONS",
            "message": "No attribute annotations were found."
        }
    ]
},
"bounding-box-metadata": {
    "objects": [
        ],
    "class-map": {
        },
    "type": "groundtruth/object-detection",
    "human-annotated": "yes",
    "creation-date": "2013-11-18 02:53:27",
    "job-name": "my job"
},
"warnings": [
    {
        "code": "WARNING_NO_ANNOTATIONS",
        "message": "No annotations were found."
    }
]
}
```

To fix **WARNING_NO_ATTRIBUTE_ANNOTATIONS**

- Choose one of the following options.
 - Add one or more bounding box annotation objects to the JSON Line. For more information, see [Object localization in manifest files \(p. 252\)](#).
 - Remove the bounding box attribute.
 - Remove the image (JSON Line) from the manifest. If other valid bounding box attributes exist in the JSON Line, you can instead remove just the invalid bounding box attribute from the JSON Line.

ERROR_UNSUPPORTED_USE_CASE_TYPE

Warning Message

More information

The value of the `type` field isn't `groundtruth/image-classification` or `groundtruth/object-detection`. For more information, see [Creating a manifest file \(p. 242\)](#).

```
{
```

```
"source-ref": "s3://bucket/test_normal_8.jpg",
"BB": [
    "annotations": [
        {
            "left": 1768,
            "top": 1007,
            "width": 448,
            "height": 295,
            "class_id": 0
        },
        {
            "left": 1794,
            "top": 1306,
            "width": 432,
            "height": 411,
            "class_id": 1
        },
        {
            "left": 2568,
            "top": 1346,
            "width": 710,
            "height": 305,
            "class_id": 2
        },
        {
            "left": 2571,
            "top": 1020,
            "width": 644,
            "height": 312,
            "class_id": 3
        }
    ],
    "image_size": [
        {
            "width": 4000,
            "height": 2667,
            "depth": 3
        }
    ]
},
"BB-metadata": {
    "job-name": "labeling-job/BB",
    "class-map": {
        "0": "comparator",
        "1": "pot_resistor",
        "2": "ir_phototransistor",
        "3": "ir_led"
    },
    "human-annotated": "yes",
    "objects": [
        {
            "confidence": 1
        },
        {
            "confidence": 1
        },
        {
            "confidence": 1
        },
        {
            "confidence": 1
        }
    ],
    "creation-date": "2021-06-22T09:58:34.811Z",
    "type": "groundtruth/wrongtype",
    "cl-errors": [

```

```
        {
            "code": "ERROR_UNSUPPORTED_USE_CASE_TYPE",
            "message": "The use case type of the BB-metadata label attribute metadata
is unsupported. Check the type field."
        }
    ],
    "cl-metadata": {
        "is_labeled": true
    },
    "cl-errors": [
        {
            "code": "ERROR_NO_VALID_LABEL_ATTRIBUTES",
            "message": "No valid label attributes found."
        }
    ]
}
```

To fix **ERROR_UNSUPPORTED_USE_CASE_TYPE**

- Choose one of the following options:
 - Change the value of the typefield to `groundtruth/image-classification` or `groundtruth/object-detection`, depending on the type of model that you want to create. For more information, see [Creating a manifest file \(p. 242\)](#).
 - Remove the image (JSON Line) from the manifest.

ERROR_INVALID_LABEL_NAME_LENGTH

More information

The length of a label name is too long. The maximum length is 256 characters.

To fix **ERROR_INVALID_LABEL_NAME_LENGTH**

- Choose one of the following options:
 - Reduce the length of the label name to 256 characters or less.
 - Remove the image (JSON Line) from the manifest.

Improving a trained Amazon Rekognition Custom Labels model

When training completes, you evaluate the performance of the model. To help you, Amazon Rekognition Custom Labels provides summary metrics and evaluation metrics for each label. For information about the available metrics, see [Metrics for evaluating your model \(p. 152\)](#). To improve your model using metrics, see [Improving an Amazon Rekognition Custom Labels model \(p. 163\)](#).

If you're satisfied with the accuracy of your model, you can start to use it. For more information, see [Running a trained Amazon Rekognition Custom Labels model \(p. 165\)](#).

Topics

- [Metrics for evaluating your model \(p. 152\)](#)
- [Accessing evaluation metrics \(Console\) \(p. 155\)](#)
- [Accessing Amazon Rekognition Custom Labels evaluation metrics \(SDK\) \(p. 156\)](#)
- [Improving an Amazon Rekognition Custom Labels model \(p. 163\)](#)

Metrics for evaluating your model

After your model is trained, Amazon Rekognition Custom Labels returns a number of metrics from model testing that you can use to evaluate the performance of your model. This topic describes the metrics available to you, and how to understand if your trained model is performing well.

The Amazon Rekognition Custom Labels console provides the following metrics as a summary of the training results and as metrics for each label:

- [Precision \(p. 153\)](#)
- [Recall \(p. 154\)](#)
- [F1 \(p. 154\)](#)

Each metric we provide is a commonly used metric for evaluating the performance of a Machine Learning model. Amazon Rekognition Custom Labels returns metrics for the results of testing across the entire test dataset, as well as metrics for each custom label. You are also able to review the performance of your trained custom model for each image in your test dataset. For more information, see [Accessing evaluation metrics \(Console\) \(p. 155\)](#).

Evaluating model performance

During testing, Amazon Rekognition Custom Labels predicts if a test image contains a custom label. The confidence score is a value that quantifies the certainty of the model's prediction.

If the confidence score for a custom label exceeds the threshold value, the model output will include this label. Predictions can be categorized in the following ways:

- *True positive* – The Amazon Rekognition Custom Labels model correctly predicts the presence of the custom label in the test image. That is, the predicted label is also a "ground truth" label for that image. For example, Amazon Rekognition Custom Labels correctly returns a soccer ball label when a soccer ball is present in an image.
- *False positive* – The Amazon Rekognition Custom Labels model incorrectly predicts the presence of a custom label in a test image. That is, the predicted label isn't a ground truth label for the image. For example, Amazon Rekognition Custom Labels returns a soccer ball label, but there is no soccer ball label in the ground truth for that image.
- *False negative* – The Amazon Rekognition Custom Labels model doesn't predict that a custom label is present in the image, but the "ground truth" for that image includes this label. For example, Amazon Rekognition Custom Labels doesn't return a 'soccer ball' custom label for an image that contains a soccer ball.
- *True negative* – The Amazon Rekognition Custom Labels model correctly predicts that a custom label isn't present in the test image. For example, Amazon Rekognition Custom Labels doesn't return a soccer ball label for an image that doesn't contain a soccer ball.

The console provides access to true positive, false positive, and false negative values for each image in your test dataset. For more information, see [Accessing evaluation metrics \(Console\) \(p. 155\)](#).

These prediction results are used to calculate the following metrics for each label, as well as an aggregate for your entire test set. The same definitions apply to predictions made by the model at the bounding box level, with the distinction that all metrics are calculated over each bounding box (prediction or ground truth) in each test image.

Intersection over Union (IoU) and object detection

Intersection over Union (IoU) measures the percentage of overlap between two object bounding boxes over their combined area. The range is 0 (lowest overlap) to 1 (complete overlap). During testing, a predicted bounding box is correct, if the IoU of the ground truth bounding box and the predicted bounding box is at least 0.5.

Assumed threshold

Amazon Rekognition Custom Labels automatically calculates an assumed threshold value (0-1) for each of your custom labels. You can't set the assumed threshold value for a custom label. The *assumed threshold* for each label is the value above which a prediction is counted as a true or false positive. It is set based on your test dataset. The assumed threshold is calculated based on the best F1 score achieved on the test dataset during model training.

You can get the value of the assumed threshold for a label from the model's training results. For more information, see [Accessing evaluation metrics \(Console\) \(p. 155\)](#).

Changes to assumed threshold values are typically used to improve the precision and recall of a model. For more information, see [Improving an Amazon Rekognition Custom Labels model \(p. 163\)](#). Since you can't set a model's assumed threshold for a label, you can achieve the same results by analyzing an image with `DetectCustomLabels` and specifying `MinConfidence` input parameter. For more information, see [Analyzing an image with a trained model \(p. 182\)](#).

Precision

Amazon Rekognition Custom Labels provides precision metrics for each label and an average precision metric for the entire test dataset.

Precision is the fraction of correct predictions (true positives) over all model predictions (true and false positives) at the assumed threshold for an individual label. As the threshold is increased, the model

might make fewer predictions. In general, however, it will have a higher ratio of true positives over false positives compared to a lower threshold. Possible values for precision range from 0–1, and higher values indicate higher precision.

For example, when the model predicts that a soccer ball is present in an image, how often is that prediction correct? Suppose there's an image with 8 soccer balls and 5 rocks. If the model predicts 9 soccer balls—8 correctly predicted and 1 false positive—then the precision for this example is 0.89. However, if the model predicted 13 soccer balls in the image with 8 correct predictions and 5 incorrect, then the resulting precision is lower.

For more information, see [Precision and recall](#).

Recall

Amazon Rekognition Custom Labels provides average recall metrics for each label and an average recall metric for the entire test dataset.

Recall is the fraction of your test set labels that were predicted correctly above the assumed threshold. It is a measure of how often the model can predict a custom label correctly when it's actually present in the images of your test set. The range for recall is 0–1. Higher values indicate a higher recall.

For example, if an image contains 8 soccer balls, how many of them are detected correctly? In the preceding example where an image has 8 soccer balls and 5 rocks, if the model detects 5 of the soccer balls, then the recall value is 0.62. If after retraining, the new model detected 9 soccer balls, including all 8 that were present in the image, then the recall value is 1.0.

For more information, see [Precision and recall](#).

F1

Amazon Rekognition Custom Labels uses the F1 score metric to measure the average model performance of each label and the average model performance of the entire test dataset.

Model performance is an aggregate measure that takes into account both precision and recall over all labels. (for example, F1 score or average precision). The model performance score is a value between 0 and 1. The higher the value, the better the model is performing for both recall and precision. Specifically, model performance for classification tasks is commonly measured by F1 score, which is the harmonic mean of the precision and recall scores at the assumed threshold. For example, for a model with precision of 0.9 and a recall of 1.0, the F1 score is 0.947.

A high value for F1 score indicates that the model is performing well for both precision and recall. If the model isn't performing well, for example, with a low precision of 0.30 and a high recall of 1.0, the F1 score is 0.46. Similarly if the precision is high (0.95) and the recall is low (0.20), the F1 score is 0.33. In both cases, the F1 score is low and indicates problems with the model.

For more information, see [F1 score](#).

Using metrics

For a given model that you have trained and depending on your application, you can make a trade-off between *precision* and *recall* by using the `MinConfidence` input parameter to `DetectCustomLabels`. At a higher `MinConfidence` value, you generally get higher *precision* (more correct predictions of soccer balls), but lower *recall* (more actual soccer balls will be missed). At a lower `MinConfidence` value, you get higher *recall* (more actual soccer balls will be correctly predicted), but lower *precision* (more of the soccer ball predictions will be wrong). For more information, see [Analyzing an image with a trained model \(p. 182\)](#).

The metrics also inform you on the steps you might take to improve model performance if needed. For more information, see [Improving an Amazon Rekognition Custom Labels model \(p. 163\)](#).

Note

`DetectCustomLabels` returns predictions ranging from 0 to 100 which correspond to the metric range of 0-1.

Accessing evaluation metrics (Console)

During testing, the model is evaluated for its performance against the test dataset. The labels in the test dataset are considered 'ground truth' as they represent what the actual image represents. During testing, the model makes predictions using the test dataset. The predicted labels are compared with the ground truth labels and the results are available in the console evaluation page.

The Amazon Rekognition Custom Labels console shows summary metrics for the entire model and metrics for individual labels. The metrics available in the console are precision, recall, F1 score, confidence, and confidence threshold. For more information, see [Improving a trained Amazon Rekognition Custom Labels model \(p. 152\)](#).

You can use the console to focus on individual metrics. For example, to investigate precision issues for a label, you can filter the training results by label and by *false positive* results. For more information, see [Metrics for evaluating your model \(p. 152\)](#).

After training, the training dataset is read-only. If you decide to improve the model, you can copy the training dataset to a new dataset. You use the copy of the dataset to train a new version of the model.

In this step, you use the console to access the training results in the console.

To access evaluation metrics (console)

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. Choose **Use Custom Labels**.
3. Choose **Get started**.
4. In the left navigation pane, choose **Projects**.
5. In the **Projects** page, choose the project that contains the trained model that you want to evaluate.
6. In the **Models** choose the model that you want to evaluate.
7. Choose the **Evaluation** tab to see the evaluation results. For information about evaluating a model, see [Improving a trained Amazon Rekognition Custom Labels model \(p. 152\)](#).
8. Choose **View test results** to see the results for individual test images. For more information, see [Metrics for evaluating your model \(p. 152\)](#).

Evaluation results

F1 score Info 0.902	Average precision Info 0.893	Overall recall Info 0.928
Date completed July 13, 2021 Trained in 1.223 hours	Training dataset 10 labels, 61 images	Testing dataset 10 labels, 56 images

Per label performance (10)

Label name	F1 score	Test images	Precision	Recall	Assumed threshold
backyard	0.857	4	1.000	0.750	0.286
bathroom	0.889	9	0.889	0.889	0.185
bedroom	0.900	11	1.000	0.818	0.262
closet	1.000	2	1.000	1.000	0.169
entry_way	1.000	3	1.000	1.000	0.149
floor_plan	1.000	2	1.000	1.000	0.685

9. After viewing the test results, choose the project name to return to the model page.

Images (56) [Info](#)

backyard2.jpeg	backyard4.jpeg
Labels front_yard False positive	Labels backyard True positive
Confidence 30.3%	Confidence 46.3%

10. Use the metrics to evaluate the performance of the model. For more information, see [Improving an Amazon Rekognition Custom Labels model \(p. 163\)](#).

Accessing Amazon Rekognition Custom Labels evaluation metrics (SDK)

The Amazon Rekognition API provides metrics beyond those provided in the console.

Like the console, the API provides access to the following metrics as summary information for the testing results and as testing results for each label:

- [Precision \(p. 153\)](#)
- [Recall \(p. 154\)](#)
- [F1 \(p. 154\)](#)

The average threshold for all labels and the threshold for individual labels is returned.

The API also provides the following metrics for classification and image detection (object location on image).

- *Confusion Matrix* for image classification.
- *Mean Average Precision (mAP)* for image detection.
- *Mean Average Recall (mAR)* for image detection.

The API also provides true positive, false positive, false negative, and true negative values. For more information, see [Metrics for evaluating your model \(p. 152\)](#).

The aggregate F1 score metric is returned directly by the API. Other metrics are accessible from a [Summary file \(p. 157\)](#) and [Evaluation manifest snapshot \(p. 158\)](#) files stored in an Amazon S3 bucket. For more information, see [Accessing the summary file and evaluation manifest snapshot \(SDK\) \(p. 161\)](#).

Topics

- [Summary file \(p. 157\)](#)
- [Evaluation manifest snapshot \(p. 158\)](#)
- [Accessing the summary file and evaluation manifest snapshot \(SDK\) \(p. 161\)](#)
- [Reference: Training results summary file \(p. 161\)](#)

Summary file

The summary file contains evaluation results information about the model as a whole and metrics for each label. The metrics are precision, recall, F1 score. The threshold value for the model is also supplied. The summary file location is accessible from the `EvaluationResult` object returned by `DescribeProjectVersions`. For more information, see [Reference: Training results summary file \(p. 161\)](#).

The following is an example summary file.

```
{  
  "Version": 1,  
  "AggregatedEvaluationResults": {  
    "ConfusionMatrix": [  
      {  
        "GroundTruthLabel": "CAP",  
        "PredictedLabel": "CAP",  
        "Value": 0.9948717948717949  
      },  
      {  
        "GroundTruthLabel": "CAP",  
        "PredictedLabel": "WATCH",  
        "Value": 0.008547008547008548  
      },  
      {  
        "GroundTruthLabel": "WATCH",  
        "PredictedLabel": "CAP",  
        "Value": 0.008547008547008548  
      }  
    ]  
  }  
}
```

```
        "Value": 0.1794871794871795
    },
    {
        "GroundTruthLabel": "WATCH",
        "PredictedLabel": "WATCH",
        "Value": 0.7008547008547008
    }
],
"F1Score": 0.9726959470546408,
"Precision": 0.9719115848331294,
"Recall": 0.9735042735042735
},
"EvaluationDetails": {
    "EvaluationEndTimestamp": "2019-11-21T07:30:23.910943",
    "Labels": [
        "CAP",
        "WATCH"
    ],
    "NumberOfTestingImages": 624,
    "NumberOfTrainingImages": 5216,
    "ProjectVersionArn": "arn:aws:rekognition:us-east-1:nnnnnnnnn:project/my-project/version/v0/1574317227432"
},
"LabelEvaluationResults": [
    {
        "Label": "CAP",
        "Metrics": {
            "F1Score": 0.9794344473007711,
            "Precision": 0.9819587628865979,
            "Recall": 0.9769230769230769,
            "Threshold": 0.9879502058029175
        },
        "NumberOfTestingImages": 390
    },
    {
        "Label": "WATCH",
        "Metrics": {
            "F1Score": 0.9659574468085106,
            "Precision": 0.961864406779661,
            "Recall": 0.9700854700854701,
            "Threshold": 0.014450683258473873
        },
        "NumberOfTestingImages": 234
    }
]
}
```

Evaluation manifest snapshot

The evaluation manifest snapshot contains detailed information about the test results. The snapshot includes the confidence rating for each prediction, and the classification of the prediction compared (true positive, true negative, false positive, or false negative) to the actual classification of the image.

The files are a snapshot since only images that could be used for testing and training are included. Images that can't be verified, such as images in the wrong format, aren't included in the manifest. The testing snapshot location is accessible from the `TestingDataResult` object returned by `DescribeProjectVersions`. The training snapshot location is accessible from `TrainingDataResult` object returned by `DescribeProjectVersions`.

The snapshot is in SageMaker Ground Truth manifest output format with fields added to provide additional information, such as the result of a detection's binary classification. The following snippet shows the additional fields.

```
"rekognition-custom-labels-evaluation-details": {  
    "version": 1,  
    "is-true-positive": true,  
    "is-true-negative": false,  
    "is-false-positive": false,  
    "is-false-negative": false,  
    "is-present-in-ground-truth": true  
    "ground-truth-labelling-jobs": ["rekognition-custom-labels-training-job"]  
}
```

- *version* – The version of the `rekognition-custom-labels-evaluation-details` field format within the manifest snapshot.
- *is-true-positive...* – The binary classification of the prediction based on how the confidence score compares to the minimum threshold for the label.
- *is-present-in-ground-truth* – True if the prediction made by the model is present in the ground truth information used for training, otherwise false. This value isn't based on whether the confidence score exceeds the minimum threshold calculated by the model.
- *ground-truth-labeling-jobs* – A list of ground truth fields in the manifest line that are used for training.

For information about the SageMaker Ground Truth manifest format, see [Output](#).

The following is an example testing manifest snapshot that shows metrics for image classification and object detection.

```
// For image classification  
{  
    "source-ref": "s3://test-bucket/dataset/beckham.jpeg",  
    "rekognition-custom-labels-training-0": 1,  
    "rekognition-custom-labels-training-0-metadata": {  
        "confidence": 1.0,  
        "job-name": "rekognition-custom-labels-training-job",  
        "class-name": "Football",  
        "human-annotated": "yes",  
        "creation-date": "2019-09-06T00:07:25.488243",  
        "type": "groundtruth/image-classification"  
    },  
    "rekognition-custom-labels-evaluation-0": 1,  
    "rekognition-custom-labels-evaluation-0-metadata": {  
        "confidence": 0.95,  
        "job-name": "rekognition-custom-labels-evaluation-job",  
        "class-name": "Football",  
        "human-annotated": "no",  
        "creation-date": "2019-09-06T00:07:25.488243",  
        "type": "groundtruth/image-classification",  
        "rekognition-custom-labels-evaluation-details": {  
            "version": 1,  
            "ground-truth-labelling-jobs": ["rekognition-custom-labels-training-job"],  
            "is-true-positive": true,  
            "is-true-negative": false,  
            "is-false-positive": false,  
            "is-false-negative": false,  
            "is-present-in-ground-truth": true  
        }  
    }  
}  
  
// For object detection  
{  
    "source-ref": "s3://test-bucket/dataset/beckham.jpeg",  
    "rekognition-custom-labels-training-0": {
```

```
"annotations": [
  {
    "class_id": 0,
    "width": 39,
    "top": 409,
    "height": 63,
    "left": 712
  },
  ...
],
"image_size": [
  {
    "width": 1024,
    "depth": 3,
    "height": 768
  }
],
"rekognition-custom-labels-training-0-metadata": {
  "job-name": "rekognition-custom-labels-training-job",
  "class-map": {
    "0": "Cap",
    ...
  },
  "human-annotated": "yes",
  "objects": [
    {
      "confidence": 1.0
    },
    ...
  ],
  "creation-date": "2019-10-21T22:02:18.432644",
  "type": "groundtruth/object-detection"
},
"rekognition-custom-labels-evaluation": {
  "annotations": [
    {
      "class_id": 0,
      "width": 39,
      "top": 409,
      "height": 63,
      "left": 712
    },
    ...
  ],
  "image_size": [
    {
      "width": 1024,
      "depth": 3,
      "height": 768
    }
  ],
  "rekognition-custom-labels-evaluation-metadata": {
    "confidence": 0.95,
    "job-name": "rekognition-custom-labels-evaluation-job",
    "class-map": {
      "0": "Cap",
      ...
    },
    "human-annotated": "no",
    "objects": [
      {
        "confidence": 0.95,
        "rekognition-custom-labels-evaluation-details": {
          "version": 1,
        }
      }
    ]
  }
}
```

```
"ground-truth-labelling-jobs": ["rekognition-custom-labels-training-job"],  
  "is-true-positive": true,  
  "is-true-negative": false,  
  "is-false-positive": false,  
  "is-false-negative": false,  
  "is-present-in-ground-truth": true  
},  
},  
...  
],  
"creation-date": "2019-10-21T22:02:18.432644",  

```

Accessing the summary file and evaluation manifest snapshot (SDK)

To get training results, you call [DescribeProjectVersions](#). For example code, see [Describing a model \(SDK\) \(p. 279\)](#).

The location of the metrics is returned in the `ProjectVersionDescription` response from `DescribeProjectVersions`.

- `EvaluationResult` – The location of the summary file.
- `TestingDataResult` – The location of the evaluation manifest snapshot used for testing.

The F1 score and summary file location are returned in `EvaluationResult`. For example:

```
"EvaluationResult": {  
    "F1Score": 1.0,  
    "Summary": {  
        "S3Object": {  
            "Bucket": "echo-dot-scans",  

```

The evaluation manifest snapshot is stored in the location specified in the `--output-config` input parameter that you specified in [Training a model \(SDK\) \(p. 109\)](#).

Note

The amount of time, in seconds, that you are billed for training is returned in `BillableTrainingTimeInSeconds`.

For information about the metrics that are returned by the Amazon Rekognition Custom Labels, see [Accessing Amazon Rekognition Custom Labels evaluation metrics \(SDK\) \(p. 156\)](#).

Reference: Training results summary

The training results summary contains metrics you can use to evaluate your model. The summary file is also used to display metrics in the console training results page. The summary file is stored in an Amazon S3 bucket after training. To get the summary file, call `DescribeProjectVersion`. For example code, see [Accessing the summary file and evaluation manifest snapshot \(SDK\) \(p. 161\)](#).

Summary file

The following JSON is the format of the summary file.

EvaluationDetails (section 3)

Overview information about the training task. This includes the ARN of the project that the model belongs to (`ProjectVersionArn`), the date and time that training finished, the version of the model that was evaluated (`EvaluationEndTimestamp`), and a list of labels detected during training (`Labels`). Also included is the number of images used for training (`NumberOfTrainingImages`) and evaluation (`NumberOfTestingImages`).

AggregatedEvaluationResults (section 1)

You can use `AggregatedEvaluationResults` to evaluate the overall performance of the trained model when used with the testing dataset. Aggregated metrics are included for `Precision`, `Recall`, and `F1Score` metrics. For object detection (the object location on an image), `AverageRecall` (mAR) and `AveragePrecision` (mAP) metrics are returned. For classification (the type of object in an image), a confusion matrix metric is returned.

LabelEvaluationResults (section 2)

You can use `labelEvaluationResults` to evaluate the performance of individual labels. The labels are sorted by the F1 score of each label. The metrics included are `Precision`, `Recall`, `F1Score`, and `Threshold` (used for classification).

The file name is formatted as follows: `EvaluationSummary-ProjectName-VersionName.json`.

```
{
  "Version": "integer",
  // section-3
  "EvaluationDetails": {
    "ProjectVersionArn": "string",
    "EvaluationEndTimestamp": "string",
    "Labels": "[string]",
    "NumberOfTrainingImages": "int",
    "NumberOfTestingImages": "int"
  },
  // section-1
  "AggregatedEvaluationResults": {
    "Metrics": {
      "Precision": "float",
      "Recall": "float",
      "F1Score": "float",
      // The following 2 fields are only applicable to object detection
      "AveragePrecision": "float",
      "AverageRecall": "float",
      // The following field is only applicable to classification
      "ConfusionMatrix": [
        {
          "GroundTruthLabel": "string",
          "PredictedLabel": "string",
          "Value": "float"
        },
        ...
      ],
    },
    // section-2
    "LabelEvaluationResults": [
      ...
    ]
  }
}
```

```
{  
  "Label": "string",  
  "NumberOfTestingImages": "int",  
  "Metrics": {  
    "Threshold": "float",  
    "Precision": "float",  
    "Recall": "float",  
    "F1Score": "float"  
  },  
  ...  
}  
]  
}
```

Improving an Amazon Rekognition Custom Labels model

The performance of machine learning models is largely dependent on factors such as the complexity and variability of your custom labels (the specific objects and scenes you are interested in), the quality and representative power of the training dataset you provide, and the model frameworks and machine learning methods used to train the model.

Amazon Rekognition Custom Labels, makes this process simpler, and no machine learning expertise is required. However, the process of building a good model often involves iterations over data and model improvements to achieve desired performance. The following is information on how to improve your model.

Data

In general, you can improve the quality of your model with larger quantities of better quality data. Use training images that clearly show the object or scene and aren't cluttered with things you don't care about. For bounding boxes around objects, use training images that show the object fully visible and not occluded by other objects.

Make sure that your training and test datasets match the type of images that you will eventually run inference on. For objects, such as logos, where you have just a few training examples, you should provide bounding boxes around the logo in your test images. These images represent or depict the scenarios in which you want to localize the object.

To add more images to a training or test dataset, see [Adding more images to a dataset \(p. 218\)](#).

Reducing false positives (better precision)

- First, check if increasing the assumed threshold lets you keep the correct predictions, while eliminating false positives. At some point, this has diminishing gains because of the trade-off between precision and recall for a given model. You can't set the assumed threshold for a label, but you can achieve the same result by specifying a high value for the `MinConfidence` input parameter to `DetectCustomLabels`. For more information, see [Analyzing an image with a trained model \(p. 182\)](#).
- You might see one or more of your custom labels of interest (A) consistently get confused with the same class of objects (but not a label that you're interested in) (B). To help, add B as an object class label to your training dataset (along with the images that you got the false positive on). Effectively, you're helping the model learn to predict B and not A through the new training images. To add images to a training dataset, see [Adding more images to a dataset \(p. 218\)](#).

- You might find that the model is confused between two of your custom labels (A and B)—the test image with label A is predicted as having label B and vice versa. In this case, first check for mislabeled images in your training and test sets. Use the dataset gallery to manage the labels assigned to a dataset. For more information, see [Managing labels \(p. 100\)](#). Also, adding more training images that reflect this confusion will help a retrained model learn to better discriminate between A and B. To add images to a training dataset, see [Adding more images to a dataset \(p. 218\)](#).

Reducing false negatives (better recall)

- Use a lower value for the assumed threshold. You can't set the assumed threshold for a label, but you can achieve the same result by specifying a lower `MinConfidence` input parameter to `DetectCustomLabels`. For more information, see [Analyzing an image with a trained model \(p. 182\)](#).
- Use better examples to model the variety of both the object and the images in which they appear.
- Split your label into two classes that are easier to learn. For example, instead of good cookies and bad cookies, you might want good cookies, burnt cookies, and broken cookies to help the model learn each unique concept better.

Running a trained Amazon Rekognition Custom Labels model

When you're satisfied with the performance of the model, you can start to use it. You can start and stop a model by using the console or the AWS SDK. The console also includes example SDK operations that you can use.

Topics

- [Inference units \(p. 165\)](#)
- [Availability Zones \(p. 167\)](#)
- [Starting an Amazon Rekognition Custom Labels model \(p. 167\)](#)
- [Stopping an Amazon Rekognition Custom Labels model \(p. 175\)](#)

Inference units

When you start your model, you specify the number of compute resources, known as an inference unit, that the model uses.

Important

You are charged for the number of hours that your model is running and for the number of inference units that your model uses while it's running, based on how you configure the running of your model. For example, if you start the model with two inference units and use the model for 8 hours, you are charged for 16 inference hours (8 hours running time * two inference units). For more information, see [Inference hours](#). If you don't explicitly [stop your model \(p. 175\)](#), you are charged even if you are not actively analyzing images with your model.

The transactions per second (TPS) that a single inference unit supports is affected by the following.

- A model that detects image-level labels (classification) generally has a higher TPS than a model that detects and localizes objects with bounding boxes (object detection).
- The complexity of the model.
- A higher resolution image requires more time for analysis.
- More objects in an image requires more time for analysis.
- Smaller images are analyzed faster than larger images.
- An image passed as image bytes is analyzed faster than first uploading the image to an Amazon S3 bucket and then referencing the uploaded image. Images passed as image bytes must be smaller than 4.0 MB. We recommend that you use image bytes for near real time processing of images and when the image size is less than 4.0 MB. For example, images captured from an IP camera.
- Processing images stored in an Amazon S3 bucket is faster than downloading the images, converting to image bytes, and then passing the image bytes for analysis.
- Analyzing an image already stored in an Amazon S3 bucket is probably faster than analyzing the same image passed as image bytes. That's especially true if the image size is larger.

If the number of calls to `DetectCustomLabels` exceeds the maximum TPS supported by the sum of inference units that a model uses, Amazon Rekognition Custom Labels returns an `ProvisionedThroughputExceededException` exception.

Managing throughput with inference units

You can increase or decrease the throughput of your model depending on the demands on your application. To increase throughput, use additional inference units. Each additional inference unit increases your processing speed by one inference unit. For information about calculating the number of inference units that you need, see [Calculate inference units for Amazon Rekognition Custom Labels and Amazon Lookout for Vision models](#). If you want to change the supported throughput of your model, you have two options:

Manually add or remove inference units

Stop (p. 175) the model and then restart (p. 167) with the required number of inference units. The disadvantage with this approach is that the model can't receive requests while it's restarting and can't be used to handle spikes in demand. Use this approach if your model has steady throughput and your use case can tolerate 10–20 minutes of downtime. An example would be if you want to batch calls to your model using a weekly schedule.

Auto-scale inference units

If your model has to accommodate spikes in demand, Amazon Rekognition Custom Labels can automatically scale the number of inference units your model uses. As demand increases, Amazon Rekognition Custom Labels adds additional inference units to the model and removes them when demand decreases.

To let Amazon Rekognition Custom Labels automatically scale inference units for a model, start (p. 167) the model and set the maximum number of inference units that it can use by using the `MaxInferenceUnits` parameter. Setting a maximum number of inference units lets you manage the cost of running the model by limiting the number of inference units available to it. If you don't specify a maximum number of units, Amazon Rekognition Custom Labels won't automatically scale your model, only using the number of inference units that you started with. For information regarding the maximum number of inference units, see [Service Quotas](#).

You can also specify a minimum number of inference units by using the `MinInferenceUnits` parameter. This lets you specify the minimum throughput for your model, where a single inference unit represents 1 hour of processing time.

Note

You can't set the maximum number of inference units with the Amazon Rekognition Custom Labels console. Instead, specify the `MaxInferenceUnits` input parameter to the `StartProjectVersion` operation.

Amazon Rekognition Custom Labels provides the following Amazon CloudWatch Logs metrics that you can use to determine the current automatic scaling status for a model.

Metric	Description
<code>DesiredInferenceUnits</code>	The number of inference units to which Amazon Rekognition Custom Labels is scaling up or down.
<code>InServiceInferenceUnits</code>	The number of inference units that the model is using.

If `DesiredInferenceUnits` = `InServiceInferenceUnits`, Amazon Rekognition Custom Labels is not currently scaling the number of inference units.

If `DesiredInferenceUnits > InServiceInferenceUnits`, Amazon Rekognition Custom Labels is scaling up to the value of `DesiredInferenceUnits`.

If `DesiredInferenceUnits < InServiceInferenceUnits`, Amazon Rekognition Custom Labels is scaling down to the value of `DesiredInferenceUnits`.

For more information regarding the metrics returned by Amazon Rekognition Custom Labels and filtering dimensions, see [CloudWatch metrics for Rekognition](#).

To find out the maximum number of inference units that you requested for a model, call `DescribeProjectsVersion` and check the `MaxInferenceUnits` field in the response. For example code, see [Describing a model \(SDK\) \(p. 279\)](#).

Availability Zones

Amazon Rekognition Custom Labels distributes inference units across multiple Availability Zones within an AWS Region to provide increased availability. For more information, see [Availability Zones](#). To help protect your production models from Availability Zone outages and inference unit failures, start your production models with at least two inference units.

If an Availability Zone outage occurs, all inference units in the Availability Zone are unavailable and model capacity is reduced. Calls to `DetectCustomLabels` are redistributed across the remaining inference units. Such calls succeed if they don't exceed the supported Transactions Per Seconds (TPS) of the remaining inference units. After AWS repairs the Availability Zone, the inference units are restarted, and full capacity is restored.

If a single inference unit fails, Amazon Rekognition Custom Labels automatically starts a new inference unit in the same Availability Zone. Model capacity is reduced until the new inference unit starts.

Starting an Amazon Rekognition Custom Labels model

You can start running an Amazon Rekognition Custom Labels model by using the console or by using the [StartProjectVersion](#) operation.

Important

You are charged for the number of hours that your model is running and for the number of inference units that your model uses while it is running. For more information, see [Running a trained Amazon Rekognition Custom Labels model \(p. 165\)](#).

Starting a model might take a few minutes to complete. To check the current status of the model readiness, check the details page for the project or use [DescribeProjectVersions](#).

After the model is started you use `DetectCustomLabels`, to analyze images using the model. For more information, see [Analyzing an image with a trained model \(p. 182\)](#). The console also provides example code to call `DetectCustomLabels`.

Topics

- [Starting an Amazon Rekognition Custom Labels model \(Console\) \(p. 168\)](#)
- [Starting an Amazon Rekognition Custom Labels model \(SDK\) \(p. 168\)](#)

Starting an Amazon Rekognition Custom Labels model (Console)

Use the following procedure to start running an Amazon Rekognition Custom Labels model with the console. You can start the model directly from the console or use the AWS SDK code provided by the console.

To start a model (console)

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. Choose **Use Custom Labels**.
3. Choose **Get started**.
4. In the left navigation pane, choose **Projects**.
5. On the **Projects** resources page, choose the project that contains the trained model that you want to start.
6. In the **Models** section, choose the model that you want to start.
7. Choose the **Use model** tab.
8. Start model using the console

In the **Start or stop model** section do the following:

1. Select the number of inference units that you want to use. For more information, see [Running a trained Amazon Rekognition Custom Labels model \(p. 165\)](#).
2. Choose **Start**.
3. In the **Start model** dialog box, choose **Start**.

Start model using the AWS SDK

In the **Use your model** section do the following:

1. Choose **API Code**.
2. Choose either **AWS CLI** or **Python**.
3. In **Start model** copy the example code.
4. Use the example code to start your model. For more information, see [Starting an Amazon Rekognition Custom Labels model \(SDK\) \(p. 168\)](#).
9. To go back to the project overview page, choose your project name at the top of the page .
10. In the **Model** section, check the status of the model. When the model status is **RUNNING**, you can use the model to analyze images. For more information, see [Analyzing an image with a trained model \(p. 182\)](#).

Starting an Amazon Rekognition Custom Labels model (SDK)

You start a model by calling the **StartProjectVersion** API and passing the Amazon Resource Name (ARN) of the model in the **ProjectVersionArn** input parameter. You also specify the number of inference units that you want to use. For more information, see [Running a trained Amazon Rekognition Custom Labels model \(p. 165\)](#).

A model might take a while to start. The Python and Java examples in this topic use waiters to wait for the model to start. A waiter is a utility method that polls for a particular state to occur. Alternatively, you can check the current status by calling [DescribeProjectVersions](#).

To start a model (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with `AmazonRekognitionFullAccess` and permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following example code to start a model.

CLI

Change the value of `project-version-arn` to the ARN of the model that you want to start. Change the value of `--min-inference-units` to the number of inference units that you want to use. Optionally, change `--max-inference-units` to the maximum number of inference units that Amazon Rekognition Custom Labels can use to automatically scale the model.

```
aws rekognition start-project-version --project-version-arn model_arn\n  --min-inference-units minimum number of units\n  --max-inference-units maximum number of units
```

Python

Supply the following command line parameters:

- `project_arn` – the ARN of the project that contains the model that you want to start.
- `model_arn` – the ARN of the model that you want to start.
- `min_inference_units` – the number of inference units that you want to use.
- (Optional) `--max_inference_units` The maximum number of inference units that Amazon Rekognition Custom Labels can use to auto-scale the model.

```
# Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.\n# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/\n# amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)\n\nimport boto3\nimport argparse\nimport logging\nfrom botocore.exceptions import ClientError\n\nlogger = logging.getLogger(__name__)\n\n\ndef get_model_status(rek_client, project_arn, model_arn):\n    """\n        Gets the current status of an Amazon Rekognition Custom Labels model\n        :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.\n        :param project_name: The name of the project that you want to use.\n        :param model_arn: The name of the model that you want the status for.\n        :return: The model status\n    """\n\n    logger.info(f"Getting status for {model_arn}.")
```

```
# extract the model version from the model arn.
version_name = (model_arn.split("version/", 1)[1]).rpartition('/')[0]

models = rek_client.describe_project_versions(ProjectArn=project_arn,
                                              VersionNames=[version_name])

for model in models['ProjectVersionDescriptions']:

    logger.info(f"Status: {model['StatusMessage']}")
    return model["Status"]

logger.exception(f"Model {model_arn} not found.")
raise Exception(f"Model {model_arn} not found.")

def start_model(rek_client, project_arn, model_arn, min_inference_units,
                max_inference_units=None):
    """
    Starts the hosting of an Amazon Rekognition Custom Labels model.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_name: The name of the project that contains the
    model that you want to start hosting.
    :param min_inference_units: The number of inference units to use for hosting.
    :param max_inference_units: The number of inference units to use for auto-
    scaling
    the model. If not supplied, auto-scaling does not happen.
    """
    try:
        # Start the model
        logger.info(f"Starting model: {model_arn}. Please wait....")

        if max_inference_units is None:
            rek_client.start_project_version(ProjectVersionArn=model_arn,
                                              MinInferenceUnits=int(min_inference_units))
        else:
            rek_client.start_project_version(ProjectVersionArn=model_arn,
                                              MinInferenceUnits=int(
                                                min_inference_units),
                                              MaxInferenceUnits=int(max_inference_units))

        # Wait for the model to be in the running state
        version_name = (model_arn.split("version/", 1)[1]).rpartition('/')[0]
        project_version_running_waiter = rek_client.get_waiter(
            'project_version_running')
        project_version_running_waiter.wait(
            ProjectArn=project_arn, VersionNames=[version_name])

        # Get the running status
        return get_model_status(rek_client, project_arn, model_arn)

    except ClientError as err:
        logger.exception(f"Client error: Problem starting model: {err}")
        raise

    print('Done...')

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

```

```
parser.add_argument(
    "project_arn", help="The ARN of the project that contains that the model
you want to start."
)
parser.add_argument(
    "model_arn", help="The ARN of the model that you want to start."
)
parser.add_argument(
    "min_inference_units", help="The minimum number of inference units to use."
)
parser.add_argument(
    "--max_inference_units", help="The maximum number of inference units to
use for auto-scaling the model.", required=False
)

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        # start the model
        rek_client = boto3.client('rekognition')
        status = start_model(rek_client,
                             args.project_arn, args.model_arn,
                             args.min_inference_units,
                             args.max_inference_units)

        print(f"Finished starting model: {args.model_arn}")
        print(f"Status: {status}")

    except ClientError as err:
        logger.exception(f"Client error: Problem starting model:{err}")
        print(f"Client error: Problem starting model: {err}")

    except Exception as err:
        logger.exception(f"Problem starting model:{err}")
        print(f"Problem starting model: {err}")

if __name__ == "__main__":
    main()
```

Java

Supply the following command line parameters:

- **project_arn** – the ARN of the project that contains the model that you want to start.
- **model_arn** – the ARN of the model that you want to start.
- **min_inference_units** – the number of inference units that you want to use.
- (Optional)**max_inference_units** – the maximum number of inference units that Amazon Rekognition Custom Labels can use to automatically scale the model. If you don't specify a value, automatic scaling doesn't happen.

```
//Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import software.amazon.awssdk.core.waiters.WaiterResponse;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsRequest;
import
software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsResponse;
import software.amazon.awssdk.services.rekognition.model.ProjectVersionDescription;
import software.amazon.awssdk.services.rekognition.model.ProjectVersionStatus;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
software.amazon.awssdk.services.rekognition.model.StartProjectVersionRequest;
import
software.amazon.awssdk.services.rekognition.model.StartProjectVersionResponse;
import software.amazon.awssdk.services.rekognition.waiters.RekognitionWaiter;

import java.net.URI;
import java.util.Optional;
import java.util.logging.Level;
import java.util.logging.Logger;

public class StartModel {

    public static final Logger logger =
Logger.getLogger(StartModel.class.getName());


    public static int findForwardSlash(String modelArn, int n) {

        int start = modelArn.indexOf('/');
        while (start >= 0 && n > 1) {
            start = modelArn.indexOf('/', start + 1);
            n -= 1;
        }
        return start;
    }

    public static void startMyModel(RekognitionClient rekClient, String projectArn,
String modelArn,
        Integer minInferenceUnits, Integer maxInferenceUnits
    ) throws Exception, RekognitionException {

        try {

            logger.log(Level.INFO, "Starting model: {0}", modelArn);

            StartProjectVersionRequest startProjectVersionRequest = null;

            if (maxInferenceUnits == null) {
                startProjectVersionRequest = StartProjectVersionRequest.builder()
                    .projectVersionArn(modelArn)
                    .minInferenceUnits(minInferenceUnits)
                    .build();
            }
            else {
                startProjectVersionRequest = StartProjectVersionRequest.builder()
                    .projectVersionArn(modelArn)
                    .minInferenceUnits(minInferenceUnits)
            }
        }
    }
}
```

```
        .maxInferenceUnits(maxInferenceUnits)
        .build();

    }

    StartProjectVersionResponse response =
rekClient.startProjectVersion(startProjectVersionRequest);

    logger.log(Level.INFO, "Status: {}", response.statusAsString() );

    // Get the model version

    int start = findForwardSlash(modelArn, 3) + 1;
    int end = findForwardSlash(modelArn, 4);

    String versionName = modelArn.substring(start, end);

    // wait until model starts

    DescribeProjectVersionsRequest describeProjectVersionsRequest =
DescribeProjectVersionsRequest.builder()
        .versionNames(versionName)
        .projectArn(projectArn)
        .build();

    RekognitionWaiter waiter = rekClient.waiter();

    WaiterResponse<DescribeProjectVersionsResponse> waiterResponse = waiter
.waitUntilProjectVersionRunning(describeProjectVersionsRequest);

    Optional<DescribeProjectVersionsResponse> optionalResponse =
waiterResponse.matched().response();

    DescribeProjectVersionsResponse describeProjectVersionsResponse =
optionalResponse.get();

    for (ProjectVersionDescription projectVersionDescription :
describeProjectVersionsResponse
        .projectVersionDescriptions()) {
        if(projectVersionDescription.status() ==
ProjectVersionStatus.RUNNING) {
            logger.log(Level.INFO, "Model is running" );
        }
        else {
            String error = "Model training failed: " +
projectVersionDescription.statusAsString() + " "
                + projectVersionDescription.statusMessage() + " " +
modelArn;
            logger.log(Level.SEVERE, error);
            throw new Exception(error);
        }
    }

} catch (RekognitionException e) {
    logger.log(Level.SEVERE, "Could not start model: {}", e.getMessage());
    throw e;
}
}
```

```
public static void main(String args[]) {  
  
    String modelArn = null;  
    String projectArn = null;  
    Integer minInferenceUnits = null;  
    Integer maxInferenceUnits = null;  
  
  
    final String USAGE = "\n" + "Usage: " + "<project_name> <version_name>  
<min_inference_units> <max_inference_units>\n\n" + "Where:\n"  
        + "    project_arn - The ARN of the project that contains the model  
        that you want to start. \n\n"  
        + "    model_arn - The ARN of the model version that you want to  
start.\n\n"  
        + "    min_inference_units - The number of inference units to start  
the model with.\n\n"  
        + "    max_inference_units - The maximum number of inference units  
that Custom Labels can use to "  
        + "    automatically scale the model. If the value is null,  
automatic scaling doesn't happen.\n\n";  
  
    if (args.length < 3 || args.length >4) {  
        System.out.println(USAGE);  
        System.exit(1);  
    }  
  
    projectArn = args[0];  
    modelArn = args[1];  
    minInferenceUnits=Integer.parseInt(args[2]);  
  
    if (args.length == 4) {  
        maxInferenceUnits = Integer.parseInt(args[3]);  
    }  
  
    try {  
  
        // Get the Rekognition client.  
        RekognitionClient rekClient = RekognitionClient.builder().build();  
  
        // Start the model.  
        startMyModel(rekClient, projectArn, modelArn, minInferenceUnits,  
maxInferenceUnits);  
  
        System.out.println(String.format("Model started: %s", modelArn));  
        rekClient.close();  
  
    } catch (RekognitionException rekError) {  
        logger.log(Level.SEVERE, "Rekognition client error: {0}",  
rekError.getMessage());  
        System.exit(1);  
    } catch (Exception rekError) {  
        logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());  
        System.exit(1);  
    }  
}  
}  
}
```

Stopping an Amazon Rekognition Custom Labels model

You can stop running an Amazon Rekognition Custom Labels model by using the console or by using the [StopProjectVersion](#) operation.

Topics

- [Stopping an Amazon Rekognition Custom Labels model \(Console\) \(p. 175\)](#)
- [Stopping an Amazon Rekognition Custom Labels model \(SDK\) \(p. 176\)](#)

Stopping an Amazon Rekognition Custom Labels model (Console)

Use the following procedure to stop a running Amazon Rekognition Custom Labels model with the console. You can stop the model directly from the console or use the AWS SDK code provided by the console.

To stop a model (console)

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. Choose **Use Custom Labels**.
3. Choose **Get started**.
4. In the left navigation pane, choose **Projects**.
5. On the **Projects** resources page, choose the project that contains the trained model that you want to stop.
6. In the **Models** section, choose the model that you want to stop.
7. Choose the **Use model** tab.
8. Stop model using the console
 1. In the **Start or stop model** section, choose **Stop**.
 2. In the **Stop model** dialog box, enter **stop** to confirm that you want to stop the model.
 3. Choose **Stop** to stop your model.

Stop model using the AWS SDK

In the **Use your model** section do the following:

1. Choose **API Code**.
2. Choose either **AWS CLI** or **Python**.
3. In **Stop model** copy the example code.
4. Use the example code to stop your model. For more information, see [Stopping an Amazon Rekognition Custom Labels model \(SDK\) \(p. 176\)](#).
9. Choose your project name at the top of the page to go back to the project overview page.
10. In the **Model** section, check the status of the model. The model has stopped when the model status is **STOPPED**.

Stopping an Amazon Rekognition Custom Labels model (SDK)

You stop a model by calling the [StopProjectVersion](#) API and passing the Amazon Resource Name (ARN) of the model in the `ProjectVersionArn` input parameter.

A model might take a while to stop. To check the current status, use `DescribeProjectVersions`.

To stop a model (SDK)

1. If you haven't already:

- a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
- b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).

2. Use the following example code to stop a running model.

CLI

Change the value of `project-version-arn` to the ARN of the model version that you want to stop.

```
aws rekognition stop-project-version --project-version-arn "my model version"
```

Python

The following example stops a model that is already running.

Supply the following command line parameters:

- `project_arn` – the ARN of the project that contains the model that you want to stop.
- `model_arn` – the ARN of the model that you want to stop.

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import argparse
import logging
import time

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def get_model_status(rek_client, project_arn, model_arn):
    """
    Gets the current status of an Amazon Rekognition Custom Labels model
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_name: The name of the project that you want to use.
    :param model_arn: The name of the model that you want the status for.
    """
    logger.info (f"Getting status for {model_arn}.")
```

```
# extract the model version from the model arn.
version_name=(model_arn.split("version/",1)[1]).rpartition('/')[0]

# get the model status
models=rek_client.describe_project_versions(ProjectArn=project_arn,
VersionNames=[version_name])

for model in models['ProjectVersionDescriptions']:

    logger.info(f"Status: {model['StatusMessage']}"))
    return model["Status"]

# no model found
logger.exception(f"Model {model_arn} not found.")
raise Exception(f"Model {model_arn} not found.")


def stop_model(rek_client, project_arn, model_arn):
    """
    Stops a running Amazon Rekognition Custom Labels Model.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_arn: The ARN of the project that you want to stop running.
    :param model_arn: The ARN of the model (ProjectVersion) that you want to stop
    running.
    """

    logger.info(f"Stopping model: {model_arn}")

    try:
        #Stop the model
        response=rek_client.stop_project_version(ProjectVersionArn=model_arn)

        logger.info(f"Status: {response['Status']}")

        # stops when hosting has stopped or failure.
        status = ""
        finished = False

        while finished is False:

            status=get_model_status(rek_client, project_arn, model_arn)

            if status == "STOPPING":
                logger.info("Model stopping in progress...")
                time.sleep(10)
                continue
            if status == "STOPPED":
                logger.info("Model is not running.")
                finished = True
                continue

        logger.exception(f"Error stopping model. Unexepeted state: {status}")
        raise Exception(f"Error stopping model. Unexepeted state: {status}")

        logger.info(f"finished. Status {status}")
        return status

    except ClientError as e:
        logger.exception(f"Couldn't stop model - {model_arn}: {e.response['Error']}"
        ['Message']))"
        raise
```

```
def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project that contains the model that
you want to stop."
    )
    parser.add_argument(
        "model_arn", help="The ARN of the model that you want to stop."
    )

def main():

    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    try:

        #get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        #stop the model
        rek_client=boto3.client('rekognition')
        status=stop_model(rek_client, args.project_arn, args.model_arn)

        print(f"Finished stopping model: {args.model_arn}")
        print(f"Status: {status}")

    except ClientError as err:
        logger.exception(f"Problem stopping model:{err}")
        print(f"Failed to stop model: {err}")

    except Exception as err:
        logger.exception(f"Problem stopping model:{err}")
        print(f"Failed to stop model: {err}")

    if __name__ == "__main__":
        main()
```

Java 2

Supply the following command line parameters:

- `project_arn` – the ARN of the project that contains the model that you want to stop.
- `model_arn` – the ARN of the model that you want to stop.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsResponse;
```

```
import software.amazon.awssdk.services.rekognition.model.ProjectVersionDescription;
import software.amazon.awssdk.services.rekognition.model.ProjectVersionStatus;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.StopProjectVersionRequest;
import
software.amazon.awssdk.services.rekognition.model.StopProjectVersionResponse;

import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;

public class StopModel {

    public static final Logger logger =
Logger.getLogger(StopModel.class.getName());

    public static int findForwardSlash(String modelArn, int n) {

        int start = modelArn.indexOf('/');
        while (start >= 0 && n > 1) {
            start = modelArn.indexOf('/', start + 1);
            n -= 1;
        }
        return start;
    }

    public static void stopMyModel(RekognitionClient rekClient, String projectArn,
String modelArn)
        throws Exception, RekognitionException {

        try {

            logger.log(Level.INFO, "Stopping {0}", modelArn);

            StopProjectVersionRequest stopProjectVersionRequest =
StopProjectVersionRequest.builder()
                .projectVersionArn(modelArn).build();

            StopProjectVersionResponse response =
rekClient.stopProjectVersion(stopProjectVersionRequest);

            logger.log(Level.INFO, "Status: {0}", response.statusAsString());

            // Get the model version

            int start = findForwardSlash(modelArn, 3) + 1;
            int end = findForwardSlash(modelArn, 4);

            String versionName = modelArn.substring(start, end);

            // wait until model stops

            DescribeProjectVersionsRequest describeProjectVersionsRequest =
DescribeProjectVersionsRequest.builder()
                .projectArn(projectArn).versionNames(versionName).build();

            boolean stopped = false;

            // Wait until create finishes

            do {

                DescribeProjectVersionsResponse describeProjectVersionsResponse =
rekClient
```

```
        .describeProjectVersions(describeProjectVersionsRequest);

        for (ProjectVersionDescription projectVersionDescription :
describeProjectVersionsResponse
            .projectVersionDescriptions()) {

            ProjectVersionStatus status =
projectVersionDescription.status();

            logger.log(Level.INFO, "stopping model: {0} ", modelArn);

            switch (status) {

                case STOPPED:
                    logger.log(Level.INFO, "Model stopped");
                    stopped = true;
                    break;

                case STOPPING:
                    Thread.sleep(5000);
                    break;

                case FAILED:
                    String error = "Model stopping failed: " +
projectVersionDescription.statusAsString() + " "
                        + projectVersionDescription.statusMessage() + " " +
modelArn;
                    logger.log(Level.SEVERE, error);
                    throw new Exception(error);

                default:
                    String unexpectedError = "Unexpected stopping state: " +
                        projectVersionDescription.statusAsString() + " "
                        + projectVersionDescription.statusMessage() + " " +
modelArn;
                    logger.log(Level.SEVERE, unexpectedError);
                    throw new Exception(unexpectedError);
            }
        }

    } while (stopped == false);

} catch (RekognitionException e) {
    logger.log(Level.SEVERE, "Could not stop model: {0}", e.getMessage());
    throw e;
}

}

public static void main(String args[]) {

    String modelArn = null;
    String projectArn = null;

    final String USAGE = "\n" + "Usage: " + "<project_name> <version_name>\n\n" +
"Where:\n" +
        "    + "    project_arn - The ARN of the project that contains the model
        that you want to stop. \n\n" +
        "    + "    model_arn - The ARN of the model version that you want to
        stop.\n\n";
    if (args.length != 2) {
        System.out.println(USAGE);
        System.exit(1);
    }
}
```

```
projectArn = args[0];
modelArn = args[1];

try {

    // Get the Rekognition client
    RekognitionClient rekClient = RekognitionClient.builder().build();

    // Stop model
    stopMyModel(rekClient, projectArn, modelArn);

    System.out.println(String.format("Model stopped: %s", modelArn));

    rekClient.close();

} catch (RekognitionException rekError) {
    logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
    System.exit(1);
} catch (Exception rekError) {
    logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());
    System.exit(1);
}

}
```

Analyzing an image with a trained model

To analyze an image with a trained Amazon Rekognition Custom Labels model, you call the [DetectCustomLabels](#) API. The result from `DetectCustomLabels` is a prediction that the image contains specific objects, scenes, or concepts.

To call `DetectCustomLabels`, you specify the following:

- The Amazon Resource Name (ARN) of the Amazon Rekognition Custom Labels model that you want to use.
- The image that you want the model to make a prediction with. You can provide an input image as an image byte array (base64-encoded image bytes), or as an Amazon S3 object. For more information, see [Image](#).

Custom labels are returned in an array of [Custom Label](#) objects. Each custom label represents a single object, scene, or concept found in the image. A custom label includes:

- A label for the object, scene, or concept found in the image.
- A bounding box for objects found in the image. The bounding box coordinates show where the object is located on the source image. The coordinate values are a ratio of the overall image size. For more information, see [BoundingBox](#). `DetectCustomLabels` returns bounding boxes only if the model is trained to detect object locations.
- The confidence that Amazon Rekognition Custom Labels has in the accuracy of the label and bounding box.

To filter labels based on the detection confidence, specify a value for `MinConfidence` that matches your desired confidence level. For example, if you need to be very confident of the prediction, specify a high value for `MinConfidence`. To get all labels, regardless of confidence, specify a `MinConfidence` value of 0.

The performance of your model is measured, in part, by the recall and precision metrics calculated during model training. For more information, see [Metrics for evaluating your model \(p. 152\)](#).

To increase the precision of your model, set a higher value for `MinConfidence`. For more information, see [Reducing false positives \(better precision\) \(p. 163\)](#).

To increase the recall of your model, use a lower value for `MinConfidence`. For more information, see [Reducing false negatives \(better recall\) \(p. 164\)](#).

If you don't specify a value for `MinConfidence`, Amazon Rekognition Custom Labels returns a label based on the assumed threshold for that label. For more information, see [Assumed threshold \(p. 153\)](#). You can get the value of the assumed threshold for a label from the model's training results. For more information, see [Training a model \(Console\) \(p. 107\)](#).

By using the `MinConfidence` input parameter, you are specifying a desired threshold for the call. Labels detected with a confidence below the value of `MinConfidence` aren't returned in the response. Also, the assumed threshold for a label doesn't affect the inclusion of the label in the response.

Note

Amazon Rekognition Custom Labels metrics express an assumed threshold as a floating point value between 0-1. The range of `MinConfidence` normalizes the threshold to a percentage value (0-100). Confidence responses from `DetectCustomLabels` are also returned as a percentage.

You might want to specify a threshold for specific labels. For example, when the precision metric is acceptable for Label A, but not for Label B. When specifying a different threshold (`MinConfidence`), consider the following.

- If you're only interested in a single label (A), set the value of `MinConfidence` to the desired threshold value. In the response, predictions for label A are returned (along with other labels) only if the confidence is greater than `MinConfidence`. You need to filter out any other labels that are returned.
- If you want to apply different thresholds to multiple labels, do the following:
 1. Use a value of 0 for `MinConfidence`. A value 0 ensures that all labels are returned, regardless of the detection confidence.
 2. For each label returned, apply the desired threshold by checking that the label confidence is greater than the threshold that you want for the label.

For more information, see [Improving a trained Amazon Rekognition Custom Labels model \(p. 152\)](#).

If you're finding the confidence values returned by `DetectCustomLabels` are too low, consider retraining the model. For more information, see [Training an Amazon Rekognition Custom Labels model \(p. 106\)](#). You can restrict the number of custom labels returned from `DetectCustomLabels` by specifying the `MaxResults` input parameter. The results are returned sorted from the highest confidence to the lowest.

For other examples that call `DetectCustomLabels`, see [Examples \(p. 309\)](#).

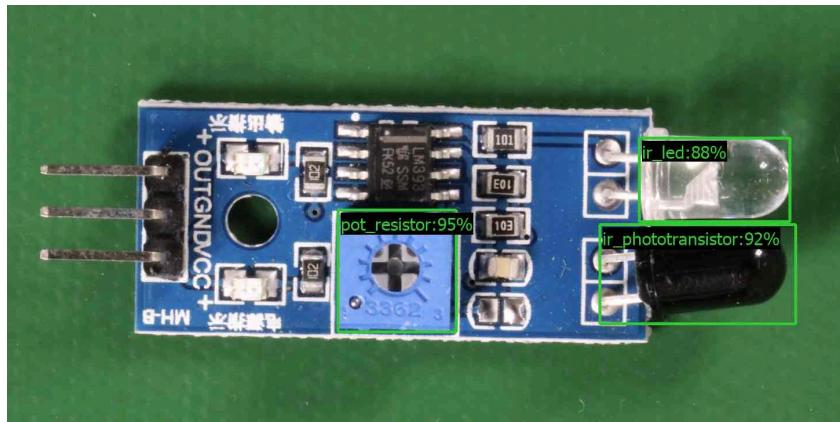
For information about securing `DetectCustomLabels`, see [Securing DetectCustomLabels \(p. 320\)](#).

To detect custom labels (API)

1. If you haven't already:
 - a. Create or update an IAM user with `AmazonRekognitionFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Train and deploy your model. For more information, see [Creating an Amazon Rekognition Custom Labels model \(p. 50\)](#).
3. Ensure the IAM user calling `DetectCustomLabels` has access to the model you used in step 2. For more information, see [Securing DetectCustomLabels \(p. 320\)](#).
4. Upload an image that you want to analyze to an S3 bucket.

For instructions, see [Uploading Objects into Amazon S3](#) in the *Amazon Simple Storage Service User Guide*. The Python, Java, and Java 2 examples also show you how to use a local image file to pass an image by using raw bytes. The file must be smaller than 4 MB.

5. Use the following examples to call the `DetectCustomLabels` operation. The Python and Java examples show the image and overlay the analysis results, similar to the following image.



AWS CLI

This AWS CLI command displays the JSON output for the DetectCustomLabels CLI operation. Change the values of the following input parameters.

- bucket with the name of Amazon S3 bucket that you used in step 4.
- image with the name of the input image file you uploaded in step 4.
- projectVersionArn with the ARN of the model that you want to use.

```
aws rekognition detect-custom-labels --project-version-arn "model_arn" \
--image '{"S3Object":{"Bucket": "bucket", "Name": "image"} }' \
--min-confidence 70
```

Python

The following example code displays bounding boxes and image level labels found in an image.

To analyze a local image, run the program and supply the following command line arguments:

- The ARN of the model with which you want to analyze the image.
- The name and location of a local image file.

To analyze an image stored in an Amazon S3 bucket, run the program and supply the following command line arguments:

- The ARN of the model with which you want to analyze the image.
- The name and location of an image within the Amazon S3 bucket that you used in step 4.
- **--bucket *bucket name*** — The Amazon S3 bucket that you used in step 4.

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)
"""

Purpose
Amazon Rekognition Custom Labels detection example used in the service
documentation:
https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/detecting-custom-
labels.html
```

```
Shows how to detect custom labels by using an Amazon Rekognition Custom Labels
model.
The image can be stored on your local computer or in an Amazon S3 bucket.
"""
import boto3
import io
import logging
import argparse
from PIL import Image, ImageDraw, ImageFont

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def analyze_local_image(rek_client, model, photo, min_confidence):
    """
    Analyzes an image stored as a local file.
    :param rek_client: The Amazon Rekognition Boto3 client.
    :param s3_connection: The Amazon S3 Boto3 S3 connection object.
    :param model: The ARN of the Amazon Rekognition Custom Labels model that you
    want to use.
    :param photo: The name and file path of the photo that you want to analyze.
    :param min_confidence: The desired threshold/confidence for the call.
    """

    try:
        logger.info ("Analyzing local file: %s", photo)
        image=Image.open(photo)
        image_type=Image.MIME[image.format]

        if (image_type == "image/jpeg" or image_type== "image/png") == False:
            logger.error("Invalid image type for %s", photo)
            raise ValueError(
                f"Invalid file format. Supply a jpeg or png format file: {photo}"
            )

        # get images bytes for call to detect_anomalies
        image_bytes = io.BytesIO()
        image.save(image_bytes, format=image.format)
        image_bytes = image_bytes.getvalue()

        response = rek_client.detect_custom_labels(Image={'Bytes': image_bytes},
            MinConfidence=min_confidence,
            ProjectVersionArn=model)

        show_image (image, response)
        return len(response['CustomLabels'])

    except ClientError as client_err:
        logger.error(format(client_err))
        raise
    except FileNotFoundError as file_error:
        logger.error(format (file_error))
        raise

def analyze_s3_image(rek_client,s3_connection, model,bucket,photo, min_confidence):
    """
    Analyzes an image stored in the specified S3 bucket.
    :param rek_client: The Amazon Rekognition Boto3 client.
    :param s3_connection: The Amazon S3 Boto3 S3 connection object.
    :param model: The ARN of the Amazon Rekognition Custom Labels model that you
    want to use.
    :param bucket: The name of the S3 bucket that contains the image that you want
    to analyze.
    :param photo: The name of the photo that you want to analyze.
```

```

:param min_confidence: The desired threshold/confidence for the call.
"""

try:
    #Get image from S3 bucket.

    logger.info("analyzing bucket: %s image: %s", bucket, photo)
    s3_object = s3_connection.Object(bucket, photo)
    s3_response = s3_object.get()

    stream = io.BytesIO(s3_response['Body'].read())
    image=Image.open(stream)

    image_type=Image.MIME[image.format]

    if (image_type == "image/jpeg" or image_type=="image/png") == False:
        logger.error("Invalid image type for %s", photo)
        raise ValueError(
            f"Invalid file format. Supply a jpeg or png format file: {photo}")

    img_width, img_height = image.size
    draw = ImageDraw.Draw(image)

    #Call DetectCustomLabels
    response = rek_client.detect_custom_labels(Image={'S3Object': {'Bucket': bucket, 'Name': photo}},
                                                MinConfidence=min_confidence,
                                                ProjectVersionArn=model)

    show_image (image, response)
    return len(response['CustomLabels'])

except ClientError as err:
    logger.error(format(err))
    raise

def show_image(image, response):
    """
    Displays the analyzed image and overlays analysis results
    :param image: The analyzed image
    :param response: the response from DetectCustomLabels
    """
    try:
        font_size=40
        line_width=5

        img_width, img_height = image.size
        draw = ImageDraw.Draw(image)

        # calculate and display bounding boxes for each detected custom label

        image_level_label_height = 0

        for custom_label in response['CustomLabels']:
            confidence=int(round(custom_label['Confidence'],0))
            label_text=f'{custom_label["Name"]}:{confidence}%'
            fnt = ImageFont.truetype('Tahoma.ttf', font_size)
            text_width, text_height = draw.textsize(label_text,fnt)

            logger.info(f"Label: {custom_label['Name']}")
            logger.info(f"Confidence: {confidence}%")

            # Draw bounding boxes, if present
            if 'Geometry' in custom_label:
    
```

```

box = custom_label['Geometry']['BoundingBox']
left = img_width * box['Left']
top = img_height * box['Top']
width = img_width * box['Width']
height = img_height * box['Height']

logger.info("Bounding box")
logger.info("\tLeft: {0:.0f}".format(left))
logger.info("\tTop: {0:.0f}".format(top))
logger.info("\tLabel Width: {0:.0f}".format(width))
logger.info("\tLabel Height: {0:.0f}".format(height))

points = (
    (left,top),
    (left + width, top),
    (left + width, top + height),
    (left , top + height),
    (left, top))
#Draw bounding box and label text
draw.line(points, fill="limegreen", width=line_width)
draw.rectangle([(left + line_width , top+line_width), (left +
text_width + line_width, top + line_width + text_height)],fill="black")
draw.text((left + line_width ,top +line_width), label_text,
fill="limegreen", font=fnt)

#draw image-level label text.
else:
    draw.rectangle([(10 , image_level_label_height), (text_width + 10,
image_level_label_height+text_height)],fill="black")
    draw.text((10,image_level_label_height), label_text,
fill="limegreen", font=fnt)

    image_level_label_height += text_height

image.show()

except Exception as err:
    logger.error(format(err))
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """
    parser.add_argument(
        "model_arn", help="The ARN of the model that you want to use."
    )

    parser.add_argument(
        "image", help="The path and file name of the image that you want to
analyze"
    )
    parser.add_argument(
        "--bucket", help="The bucket that contains the image. If not supplied,
image is assumed to be a local file.", required=False
    )

def main():

    try:
        logging.basicConfig(level=logging.INFO, format="%(levelname)s:
%(message)s")

```

```

#get command line arguments
parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
add_arguments(parser)
args = parser.parse_args()

label_count=0
min_confidence=50

rek_client=boto3.client('rekognition')

if args.bucket==None:
    # Analyze local image
    label_count=analyze_local_image(rek_client,
                                    args.model_arn,
                                    args.image,
                                    min_confidence)
else:
    #Analyze image in S3 bucket
    s3_connection = boto3.resource('s3')
    label_count=analyze_s3_image(rek_client,
                                s3_connection,
                                args.model_arn,
                                args.bucket,
                                args.image,
                                min_confidence)

print(f"Custom labels detected: {label_count}")

except ClientError as client_err:
    print("A service client error occurred: " +
format(client_err.response["Error"]["Message"]))

except ValueError as value_err:
    print ("A value error occurred: " + format(value_err))

except FileNotFoundError as file_error:
    print("File not found error: " + format (file_error))

except Exception as err:
    print("An error occurred: " + format(err))

if __name__ == "__main__":
    main()

```

Java

The following example code displays bounding boxes and image level labels found in an image.

To analyze a local image, run the program and supply the following command line arguments:

- The ARN of the model with which you want to analyze the image.
- The name and location of a local image file.

To analyze an image stored in an Amazon S3 bucket, run the program and supply the following command line arguments:

- The ARN of the model with which you want to analyze the image.
- The name and location of an image within the Amazon S3 bucket that you used in step 4.
- The Amazon S3 bucket that contains the image that you used in step 4.

```

//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

package com.example.rekognition;

import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.util.List;
import javax.imageio.ImageIO;
import javax.swing.*;
import java.io.FileNotFoundException;
import java.awt.font.FontRenderContext;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.io.File;
import java.io.FileInputStream;
import java.io.InputStream;
import java.nio.ByteBuffer;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;

import com.amazonaws.services.rekognition.AmazonRekognition;
import com.amazonaws.services.rekognition.AmazonRekognitionClientBuilder;

import com.amazonaws.services.rekognition.model.BoundingBox;
import com.amazonaws.services.rekognition.modelCustomLabel;
import com.amazonaws.services.rekognition.model.DetectCustomLabelsRequest;
import com.amazonaws.services.rekognition.model.DetectCustomLabelsResult;
import com.amazonaws.services.rekognition.model.Image;
import com.amazonaws.services.rekognition.model.S3Object;
import com.amazonaws.services.s3.AmazonS3;
import com.amazonaws.services.s3.AmazonS3ClientBuilder;
import com.amazonaws.services.s3.model.S3ObjectInputStream;

import com.amazonaws.services.rekognition.model.AmazonRekognitionException;
import com.amazonaws.services.s3.model.AmazonS3Exception;
import com.amazonaws.util.IOUtils;

// Calls DetectCustomLabels and displays a bounding box around each detected image.
public class DetectCustomLabels extends JPanel {

    private transient DetectCustomLabelsResult response;
    private transient Dimension dimension;
    private transient BufferedImage image;

    public static final Logger logger =
    Logger.getLogger(DetectCustomLabels.class.getName());

    // Finds custom labels in an image stored in an S3 bucket.
    public DetectCustomLabels(AmazonRekognition rekClient,
        AmazonS3 s3Client,
        String projectVersionArn,
        String bucket,
        String key,
        Float minConfidence) throws AmazonRekognitionException,
    AmazonS3Exception, IOException {

        logger.log(Level.INFO, "Processing S3 bucket: {0} image {1}", new Object[]
        { bucket, key });

        // Get image from S3 bucket and create BufferedImage

```

```

        com.amazonaws.services.s3.model.S3Object s3object =
s3client.getObject(bucket, key);
S3ObjectInputStream inputStream = s3object.getObjectContent();
image = ImageIO.read(inputStream);

// Set image size
setWindowDimensions();

DetectCustomLabelsRequest request = new DetectCustomLabelsRequest()
    .withProjectVersionArn(projectVersionArn)
    .withImage(new Image().withS3Object(new
S3Object().withName(key).withBucket(bucket)))
    .withMinConfidence(minConfidence);

// Call DetectCustomLabels

response = rekClient.detectCustomLabels(request);
logFoundLabels(response.getCustomLabels());
drawLabels();

}

// Finds custom label in a local image file.
public DetectCustomLabels(AmazonRekognition rekClient,
    String projectVersionArn,
    String photo,
    Float minConfidence)
    throws IOException, AmazonRekognitionException {

logger.log(Level.INFO, "Processing local file: {0}", photo);

// Get image bytes and buffered image
ByteBuffer imageBytes;
try (InputStream inputStream = new FileInputStream(new File(photo))) {
    imageBytes = ByteBuffer.wrap(IOUtils.toByteArray(inputStream));
}

// Get image for display
InputStream imageBytesStream;
imageBytesStream = new ByteArrayInputStream(imageBytes.array());

ByteArrayOutputStream baos = new ByteArrayOutputStream();
image = ImageIO.read(imageBytesStream);
ImageIO.write(image, "jpg", baos);

// Set image size
setWindowDimensions();

// Analyze image
DetectCustomLabelsRequest request = new DetectCustomLabelsRequest()
    .withProjectVersionArn(projectVersionArn)
    .withImage(new Image()
        .withBytes(imageBytes))
    .withMinConfidence(minConfidence);

response = rekClient.detectCustomLabels(request);

logFoundLabels(response.getCustomLabels());

drawLabels();

}

// Log the labels found by DetectCustomLabels
private void logFoundLabels(List<CustomLabel> customLabels) {
    logger.info("Custom labels found");
}

```

```

        if (customLabels.isEmpty()) {
            logger.log(Level.INFO, "No Custom Labels found. Consider lowering min
confidence.");
        } else {
            for (CustomLabel customLabel : customLabels) {
                logger.log(Level.INFO, " Label: {0} Confidence: {1}",
                new Object[] { customLabel.getName(),
                customLabel.getConfidence()});
            }
        }
    }

    // Sets window dimensions to 1/2 screen size, unless image is smaller
    public void setWindowDimensions() {
        dimension = java.awt.Toolkit.getDefaultToolkit().getScreenSize();

        dimension.width = (int) dimension.getWidth() / 2;
        if (image.getWidth() < dimension.width) {
            dimension.width = image.getWidth();
        }
        dimension.height = (int) dimension.getHeight() / 2;

        if (image.getHeight() < dimension.height) {
            dimension.height = image.getHeight();
        }

        setPreferredSize(dimension);
    }

    // Draws the image containing the bounding boxes and labels.
    @Override
    public void paintComponent(Graphics g) {

        Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

        // Draw the image.
        g2d.drawImage(image, 0, 0, dimension.width, dimension.height, this);
    }

    public void drawLabels() {
        // Draws bounding boxes (if present) and label text.

        int boundingBoxBorderWidth = 5;
        int imageHeight = image.getHeight(this);
        int imageWidth = image.getWidth(this);

        // Set up drawing
        Graphics2D g2d = image.createGraphics();
        g2d.setColor(Color.GREEN);
        g2d.setFont(new Font("Tahoma", Font.PLAIN, 50));
        Font font = g2d.getFont();
        FontRenderContext frc = g2d.getFontRenderContext();
        g2d.setStroke(new BasicStroke(boundingBoxBorderWidth));

        List<CustomLabel> customLabels = response.getCustomLabels();

        int imageLevelLabelHeight = 0;
        for (CustomLabel customLabel : customLabels) {

            String label = customLabel.getName();

            int textWidth = (int) (font.getStringBounds(label, frc).getWidth());
            int textHeight = (int) (font.getStringBounds(label, frc).getHeight());
        }
    }
}

```

```

// Draw bounding box, if present
if (CustomLabel.getGeometry() != null) {

    BoundingBox box = customLabel.getGeometry().getBoundingBox();
    float left = imageWidth * box.getLeft();
    float top = imageHeight * box.getTop();

    // Draw black rectangle
    g2d.setColor(Color.BLACK);
    g2d.fillRect(Math.round(left + (boundingBoxBorderWidth)),
    Math.round(top + (boundingBoxBorderWidth)),
    textWidth + boundingBoxBorderWidth, textHeight +
    boundingBoxBorderWidth);

    // Write label onto black rectangle
    g2d.setColor(Color.GREEN);
    g2d.drawString(label, left + boundingBoxBorderWidth, (top +
    textHeight));

    // Draw bounding box around label location
    g2d.drawRect(Math.round(left), Math.round(top),
    Math.round((imageWidth * box.getWidth())),
    Math.round((imageHeight * box.getHeight())));
}

// Draw image level labels.
else {
    // Draw black rectangle
    g2d.setColor(Color.BLACK);
    g2d.fillRect(10, 10 + imageLevelLabelHeight, textWidth,
    textHeight);
    g2d.setColor(Color.GREEN);
    g2d.drawString(label, 10, textHeight + imageLevelLabelHeight);

    imageLevelLabelHeight += textHeight;
}

g2d.dispose();
}

public static void main(String args[]) throws Exception {

    String photo = null;
    String bucket = null;
    String projectVersionArn = null;
    float minConfidence = 50;

    final String USAGE = "\n" + "Usage: " + "<model_arn> <image> <bucket>\n\n" +
    "Where:\n" +
    "    + " model_arn - The ARN of the model that you want to use. \n\n" +
    "    + " image - The location of the image on your local file system" +
    " or within an S3 bucket.\n\n" +
    "    + " bucket - The S3 bucket that contains the image. Don't specify" +
    " if image is local.\n\n";

    // Collect the arguments. If 3 arguments are present, the image is assumed
    to be
    // in an S3 bucket.

    if (args.length < 2 || args.length > 3) {
        System.out.println(USAGE);
        System.exit(1);
    }
}

```

```

projectVersionArn = args[0];
photo = args[1];

if (args.length == 3) {
    bucket = args[2];
}

DetectCustomLabels panel = null;

try {
    // Get the Rekognition client
    AmazonRekognition rekClient =
    AmazonRekognitionClientBuilder.defaultClient();

    AmazonS3 s3client = AmazonS3ClientBuilder.defaultClient();

    // Create frame and panel.
    JFrame frame = new JFrame("Custom Labels");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    if (args.length == 2) {
        // Analyze local image
        panel = new DetectCustomLabels(rekClient, projectVersionArn, photo,
minConfidence);
    } else {
        // Analyze image in S3 bucket
        panel = new DetectCustomLabels(rekClient, s3client,
projectVersionArn, bucket, photo, minConfidence);
    }

    frame.setContentPane(panel);
    frame.pack();
    frame.setVisible(true);

} catch (AmazonRekognitionException rekError) {
    String errorMessage = "Rekognition client error: " +
rekError.getMessage();
    logger.log(Level.SEVERE, errorMessage);
    System.out.println(errorMessage);
    System.exit(1);
} catch (FileNotFoundException fileError) {
    String errorMessage = "File not found: " + photo;
    logger.log(Level.SEVERE, errorMessage);
    System.out.println(errorMessage);
    System.exit(1);
} catch (IOException fileError) {
    String errorMessage = "Input output exception: " +
fileError.getMessage();
    logger.log(Level.SEVERE, errorMessage);
    System.out.println(errorMessage);
    System.exit(1);
} catch (AmazonS3Exception s3Error) {
    String errorMessage = "S3 error: " + s3Error.getErrorMessage();
    logger.log(Level.SEVERE, errorMessage);
    System.out.println(errorMessage);
    System.exit(1);
}
}
}

```

Java 2

The following example code displays bounding boxes and image level labels found in an image.

To analyze a local image, run the program and supply the following command line arguments:

- `projectVersionArn` – The ARN of the model with which you want to analyze the image.
- `photo` – the name and location of a local image file.

To analyze an image stored in an S3 bucket, run the program and supply the following command line arguments:

- The ARN of the model with which you want to analyze the image.
- The name and location of an image within the S3 bucket that you used in step 4.
- The Amazon S3 bucket that contains the image that you used in step 4.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import software.amazon.awssdk.core.ResponseBytes;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.core.sync.ResponseTransformer;

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.S3Object;
import software.amazon.awssdk.services.rekognition.model.Image;
import software.amazon.awssdk.services.rekognition.model.DetectCustomLabelsRequest;
import
software.amazon.awssdk.services.rekognition.model.DetectCustomLabelsResponse;
import software.amazon.awssdk.services.rekognition.modelCustomLabel;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import software.amazon.awssdk.services.rekognition.model.BoundingBox;

import software.amazon.awssdk.services.s3.S3Client;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import software.amazon.awssdk.services.s3.model.NoSuchBucketException;
import software.amazon.awssdk.services.s3.model.NoSuchKeyException;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.util.List;

import java.awt.*;
import java.awt.font.FontRenderContext;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import javax.swing.*;

import java.util.logging.Level;
import java.util.logging.Logger;

// Calls DetectCustomLabels on an image. Displays bounding boxes or
// image level labels found in the image.
public class DetectCustomLabels extends JPanel {

    private transient BufferedImage image;
    private transient DetectCustomLabelsResponse response;
    private transient Dimension dimension;
```

```

    public static final Logger logger =
Logger.getLogger(DetectCustomLabels.class.getName());

    // Finds custom labels in an image stored in an S3 bucket.
    public DetectCustomLabels(RekognitionClient rekClient,
        S3Client s3client,
        String projectVersionArn,
        String bucket,
        String key,
        Float minConfidence) throws RekognitionException,
NoSuchBucketException, NoSuchKeyException, IOException {

        logger.log(Level.INFO, "Processing S3 bucket: {0} image {1}", new Object[]
{ bucket, key });
        // Get image from S3 bucket and create BufferedImage
        GetObjectRequest requestObject =
GetObjectRequest.builder().bucket(bucket).key(key).build();
        ResponseBytes<GetObjectResponse> result = s3client.getObject(requestObject,
ResponseTransformer.toBytes());
        ByteArrayInputStream bis = new ByteArrayInputStream(result.asByteArray());
        image = ImageIO.read(bis);

        // Set image size
        setWindowDimensions();

        // Construct request parameter for DetectCustomLabels
        S3Object s3Object = S3Object.builder().bucket(bucket).name(key).build();

        Image s3Image = Image.builder().s3Object(s3Object).build();

        DetectCustomLabelsRequest request =
DetectCustomLabelsRequest.builder().image(s3Image)
.projectVersionArn(projectVersionArn).minConfidence(minConfidence).build();

        response = rekClient.detectCustomLabels(request);
        logFoundLabels(response.customLabels());
        drawLabels();

    }

    // Finds custom label in a local image file.
    public DetectCustomLabels(RekognitionClient rekClient,
        String projectVersionArn,
        String photo,
        Float minConfidence)
        throws IOException, RekognitionException {

        logger.log(Level.INFO, "Processing local file: {0}", photo);
        // Get image bytes and buffered image
        InputStream sourceStream = new FileInputStream(new File(photo));
        SdkBytes imageBytes = SdkBytes.fromInputStream(sourceStream);
        ByteArrayInputStream inputStream = new
ByteArrayInputStream(imageBytes.asByteArray());
        image = ImageIO.read(inputStream);

        setWindowDimensions();

        // Construct request parameter for DetectCustomLabels
        Image localImageBytes = Image.builder().bytes(imageBytes).build();

        DetectCustomLabelsRequest request =
DetectCustomLabelsRequest.builder().image(localImageBytes)
.projectVersionArn(projectVersionArn).minConfidence(minConfidence).build();

```

```
response = rekClient.detectCustomLabels(request);

logFoundLabels(response.customLabels());
drawLabels();

}

// Sets window dimensions to 1/2 screen size, unless image is smaller
public void setWindowDimensions() {
    dimension = java.awt.Toolkit.getDefaultToolkit().getScreenSize();

    dimension.width = (int) dimension.getWidth() / 2;
    if (image.getWidth() < dimension.width) {
        dimension.width = image.getWidth();
    }
    dimension.height = (int) dimension.getHeight() / 2;

    if (image.getHeight() < dimension.height) {
        dimension.height = image.getHeight();
    }
    setPreferredSize(dimension);
}

// Draws bounding boxes (if present) and label text.
public void drawLabels() {

    int boundingBoxBorderWidth = 5;
    int imageHeight = image.getHeight(this);
    int imageWidth = image.getWidth(this);

    // Set up drawing
    Graphics2D g2d = image.createGraphics();
    g2d.setColor(Color.GREEN);
    g2d.setFont(new Font("Tahoma", Font.PLAIN, 50));
    Font font = g2d.getFont();
    FontRenderContext frc = g2d.getFontRenderContext();
    g2d.setStroke(new BasicStroke(boundingBoxBorderWidth));

    List<CustomLabel> customLabels = response.customLabels();

    int imageLevelLabelHeight = 0;
    for (CustomLabel customLabel : customLabels) {

        String label = customLabel.name();

        int textWidth = (int) (font.getStringBounds(label, frc).getWidth());
        int textHeight = (int) (font.getStringBounds(label, frc).getHeight());

        // Draw bounding box, if present
        if (customLabel.geometry() != null) {

            BoundingBox box = customLabel.geometry().boundingBox();
            float left = imageWidth * box.left();
            float top = imageHeight * box.top();

            // Draw black rectangle
            g2d.setColor(Color.BLACK);
            g2d.fillRect(Math.round(left + (boundingBoxBorderWidth)),
                        Math.round(top + (boundingBoxBorderWidth)),
                        textWidth + boundingBoxBorderWidth, textHeight +
                        boundingBoxBorderWidth);

            // Write label onto black rectangle
            g2d.setColor(Color.GREEN);
        }
    }
}
```

```

        g2d.drawString(label, left + boundingBoxBorderWidth, (top +
textHeight));

        // Draw bounding box around label location
        g2d.drawRect(Math.round(left), Math.round(top),
Math.round((imageWidth * box.width())),
Math.round((imageHeight * box.height())));
    }
    // Draw image level labels.
    else {
        // Draw black rectangle
        g2d.setColor(Color.BLACK);
        g2d.fillRect(10, 10 + imageLevelLabelHeight, textWidth,
textHeight);
        g2d.setColor(Color.GREEN);
        g2d.drawString(label, 10, textHeight + imageLevelLabelHeight);

        imageLevelLabelHeight += textHeight;
    }
}

g2d.dispose();

}

// Log the labels found by DetectCustomLabels
private void logFoundLabels(List<CustomLabel> customLabels) {
    logger.info("Custom labels found:");
    if (customLabels.isEmpty()) {
        logger.log(Level.INFO, "No Custom Labels found. Consider lowering min
confidence.");
    }
    else {
        for (CustomLabel customLabel : customLabels) {
            logger.log(Level.INFO, " Label: {0} Confidence: {1}",
new Object[] { customLabel.name(),
customLabel.confidence() } );
        }
    }
}

// Draws the image containing the bounding boxes and labels.
@Override
public void paintComponent(Graphics g) {

    Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

    // Draw the image.
    g2d.drawImage(image, 0, 0, dimension.width, dimension.height, this);

}

public static void main(String args[]) throws Exception {

    String photo = null;
    String bucket = null;
    String projectVersionArn = null;

    final String USAGE = "\n" + "Usage: " + "<model_arn> <image> <bucket>\n\n"
+ "Where:\n"
        + "    model_arn - The ARN of the model that you want to use. \n\n"
        + "    image - The location of the image on your local file system
or within an S3 bucket.\n\n"
        + "    bucket - The S3 bucket that contains the image. Don't specify
if image is local.\n\n";
}

```

```

    // Collect the arguments. If 3 arguments are present, the image is assumed
    to be
    // in an S3 bucket.

    if (args.length < 2 || args.length > 3) {
        System.out.println(USAGE);
        System.exit(1);
    }

    projectVersionArn = args[0];
    photo = args[1];

    if (args.length == 3) {
        bucket = args[2];
    }

    float minConfidence = 50;

    DetectCustomLabels panel = null;

    try {
        // Get the Rekognition client
        RekognitionClient rekClient = RekognitionClient.builder().build();

        S3Client s3client = S3Client.builder().build();

        // Create frame and panel.
        JFrame frame = new JFrame("Custom Labels");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        if (args.length == 2) {
            // Analyze local image
            panel = new DetectCustomLabels(rekClient, projectVersionArn, photo,
minConfidence);
        } else {
            // Analyze image in S3 bucket
            panel = new DetectCustomLabels(rekClient, s3client,
projectVersionArn, bucket, photo, minConfidence);
        }

        frame.setContentPane(panel);
        frame.pack();
        frame.setVisible(true);

    } catch (RekognitionException rekError) {

        String errorMessage = "Rekognition client error: " +
rekError.getMessage();
        logger.log(Level.SEVERE, errorMessage);
        System.out.println(errorMessage);
        System.exit(1);
    } catch (FileNotFoundException fileError) {
        String errorMessage = "File not found: " + photo;
        logger.log(Level.SEVERE, errorMessage);
        System.out.println(errorMessage);
        System.exit(1);
    } catch (IOException fileError) {
        String errorMessage = "Input output exception: " +
fileError.getMessage();
        logger.log(Level.SEVERE, errorMessage);
        System.out.println(errorMessage);
        System.exit(1);
    } catch (NoSuchKeyException bucketError) {
        String errorMessage = String.format("Image not found: %s in bucket
%s.", photo, bucket);
    }
}

```

```
        logger.log(Level.SEVERE, errorMessage);
        System.out.println(errorMessage);
        System.exit(1);
    } catch (NoSuchBucketException bucketError) {
        String errorMessage = "Bucket not found: " + bucket;
        logger.log(Level.SEVERE, errorMessage);
        System.out.println(errorMessage);
        System.exit(1);
    }
}
```

DetectCustomLabels operation request

In the DetectCustomLabels operation, you supply an input image either as a base64-encoded byte array or as an image stored in an Amazon S3 bucket. The following example JSON request shows the image loaded from an Amazon S3 bucket.

```
{
    "ProjectVersionArn": "string",
    "Image": {
        "S3Object": {
            "Bucket": "string",
            "Name": "string",
            "Version": "string"
        }
    },
    "MinConfidence": 90,
    "MaxLabels": 10,
}
```

DetectCustomLabels operation response

The following JSON response from the DetectCustomLabels operation shows the custom labels that were detected in the following image.

```
{
    "CustomLabels": [
        {
            "Name": "MyLogo",
            "Confidence": 77.7729721069336,
            "Geometry": {
                "BoundingBox": {
                    "Width": 0.198987677693367,
                    "Height": 0.31296101212501526,
                    "Left": 0.07924537360668182,
                    "Top": 0.4037395715713501
                }
            }
        }
    ]
}
```

Managing Amazon Rekognition Custom Labels resources

This section gives you an overview of the workflow you use to train and use an Amazon Rekognition Custom Labels model. Also included is overview information for using the AWS SDK to train and use a model.

Managing an Amazon Rekognition Custom Labels project

Within Amazon Rekognition Custom Labels, you use a project to manage the models that you create for a specific use case. A project manages datasets, model training, model versions, model evaluation, and the running of your project's models.

Topics

- [Deleting an Amazon Rekognition Custom Labels project \(p. 200\)](#)
- [Describing a project \(SDK\) \(p. 207\)](#)
- [Creating a project with AWS CloudFormation \(p. 211\)](#)

Deleting an Amazon Rekognition Custom Labels project

You can delete a project by using the Amazon Rekognition console or by calling the [DeleteProject API](#). To delete a project, you must first delete each associated model. A deleted project or model can't be undeleted.

Topics

- [Deleting an Amazon Rekognition Custom Labels project \(Console\) \(p. 200\)](#)
- [Deleting an Amazon Rekognition Custom Labels project \(SDK\) \(p. 202\)](#)

Deleting an Amazon Rekognition Custom Labels project (Console)

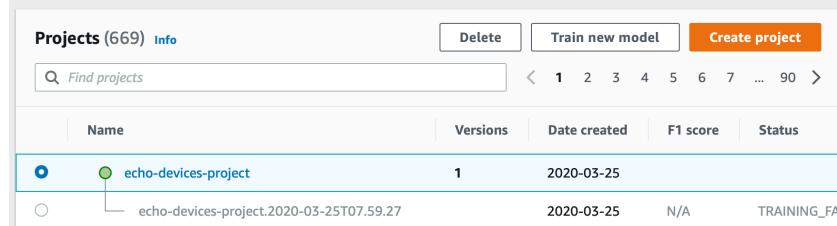
You can delete a project from the projects page, or you can delete a project from a project's detail page. The following procedure shows you how to delete a project using the projects page.

The Amazon Rekognition Custom Labels console deletes associated models and datasets for you during project deletion. You can't delete a project if any of its models are running or training. To stop a running model, see [Stopping an Amazon Rekognition Custom Labels model \(SDK\) \(p. 176\)](#). If a model is training, wait until it finishes before you delete the project.

To delete a project (console)

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.

2. Choose **Use Custom Labels**.
3. Choose **Get started**.
4. In the left navigation pane, choose **Projects**.
5. On the **Projects** page, select the radio button for the project that you want to delete.



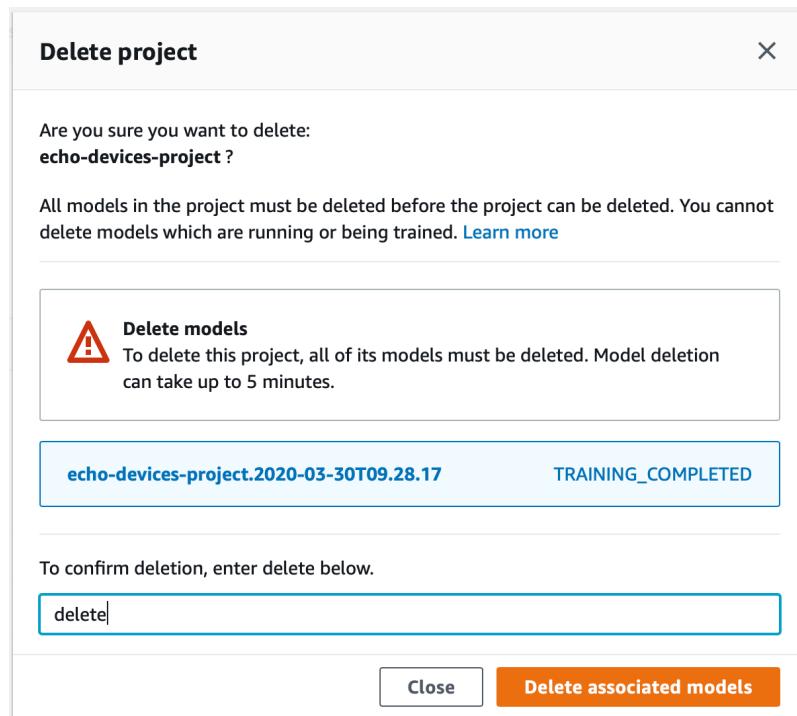
Name	Versions	Date created	F1 score	Status
echo-devices-project	1	2020-03-25		
echo-devices-project.2020-03-25T07.59.27		2020-03-25	N/A	TRAINING_FA

6. Choose **Delete** at the top of the page. The **Delete project** dialog box is shown.
7. If the project has no associated models:
 - a. Enter **delete** to delete the project.
 - b. Choose **Delete** to delete the project.
8. If the project has associated models or datasets:
 - a. Enter **delete** to confirm that you want to delete the model(s) and datasets.
 - b. Choose either **Delete associated models** or **Delete associated datasets** or **Delete associated datasets and models**, depending on whether the model has datasets, models, or both. Model deletion might take a while to complete.

Note

The console can't delete models that are in-training or running. Try again after stopping any running models that are listed, and wait until models listed as training finish.

If you **Close** the dialog box during model deletion, the models are still deleted. Later, you can delete the project by repeating this procedure.



Are you sure you want to delete:
echo-devices-project?

All models in the project must be deleted before the project can be deleted. You cannot delete models which are running or being trained. [Learn more](#)

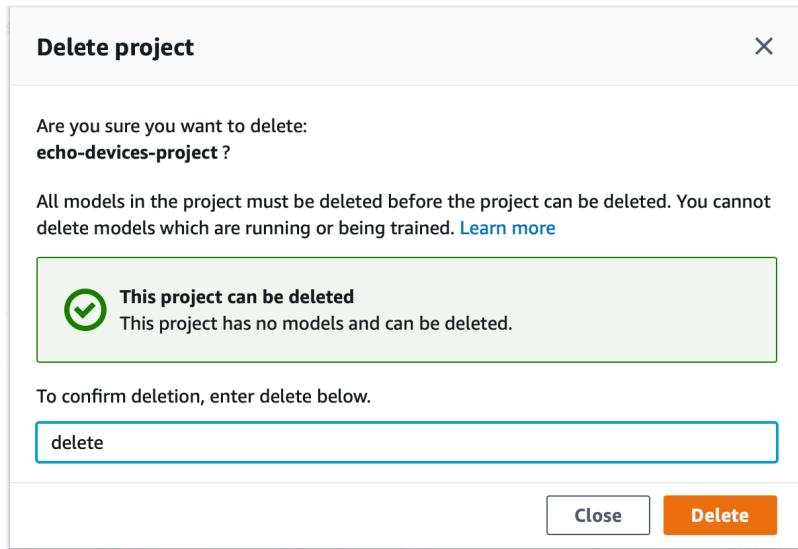
Delete models
To delete this project, all of its models must be deleted. Model deletion can take up to 5 minutes.

echo-devices-project.2020-03-30T09.28.17	TRAINING_COMPLETED
---	--------------------

To confirm deletion, enter delete below.

Close **Delete associated models**

- c. Enter **delete** to confirm that you want to delete the project.
- d. Choose **Delete** to delete the project.



Deleting an Amazon Rekognition Custom Labels project (SDK)

You delete an Amazon Rekognition Custom Labels project by calling `DeleteProject` and supplying the Amazon Resource Name (ARN) of the project that you want to delete. To get the ARNs of the projects in your account, call `DescribeProjects`. The response includes an array of `ProjectDescription` objects. The project ARN is the `ProjectArn` field. You can use the project name to identify the ARN of the project. For example, `arn:aws:rekognition:us-east-1:123456789010:project/project name/1234567890123`.

Before you can delete a project, you must first delete all models and datasets in the project. For more information, see [Deleting an Amazon Rekognition Custom Labels model \(SDK\) \(p. 269\)](#) and [Deleting a dataset \(p. 238\)](#).

The project might take a few moments to delete. During that time, the status of the project is `DELETING`. The project is deleted if a subsequent call to `DescribeProjects` doesn't include the project that you deleted.

To delete a project (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following code to delete a project.

AWS CLI

Change the value of `project-arn` to the name of the project that you want to delete.

```
aws rekognition delete-project --project-arn project_arn
```

Python

Use the following code. Supply the following command line parameters:

- `project_arn`— the ARN of the project that you want to delete.

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

"""
Purpose
Amazon Rekognition Custom Labels project example used in the service documentation:
https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/mp-delete-
project.html
Shows how to delete an existing Amazon Rekognition Custom Labels project.
You must first delete any models and datasets that belong to the project.
"""

import boto3
import argparse
import logging
import time
import json

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def find_forward_slash(input_string, n):
    """
    Returns the location of '/' after n number of occurrences.
    :param input_string: The string you want to search
    :param n: the occurrence that you want to find.
    """
    position = input_string.find('/')
    while position >= 0 and n > 1:
        position = input_string.find('/', position + 1)
        n -= 1
    return position

def delete_project(rek_client, project_arn):
    """
    Deletes an Amazon Rekognition Custom Labels project.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_arn: The ARN of the project that you want to delete.
    """
    try:
        #Delete the project
        logger.info(f"Deleting project: {project_arn}")

        response=rek_client.delete_project(ProjectArn=project_arn)

        logger.info(f"project status: {response['Status']}")

        deleted=False

        logger.info(f"waiting for project deletion {project_arn}")

        # Get the project name
        start=find_forward_slash(project_arn, 1) +1
```

```
end=find_forward_slash(project_arn,2)
project_name=project_arn[start:end]

project_names = [project_name]

while deleted==False:

    project_descriptions=rek_client.describe_projects(ProjectNames=project_names)
    ['ProjectDescriptions']

    if len(project_descriptions) == 0:
        deleted=True

    else:
        time.sleep(5)

    logger.info(f"project deleted: {project_arn}")

    return True

except ClientError as err:
    logger.exception(f"Couldn't delete project - {project_arn}:
{err.response['Error']['Message']}")

raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project that you want to delete."
    )

def main():
    logging.basicConfig(level=logging.INFO, format"%(levelname)s: %(message)s")
    try:

        #get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Deleting project: {args.project_arn}")

        #Delete the project
        rek_client=boto3.client('rekognition')

        delete_project(rek_client,
                      args.project_arn)

        print(f"Finished deleting project: {args.project_arn}")

    except ClientError as err:
        logger.exception(f"Problem deleting project: {err}")
        print(f"Problem deleting project: {err}")


```

```
if __name__ == "__main__":
    main()
```

Java V2

Use the following code. Supply the following command line parameters:

- `project_arn`— the ARN of the project that you want to delete.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import java.net.URI;
import java.util.List;
import java.util.Objects;
import java.util.logging.Level;
import java.util.logging.Logger;

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DeleteProjectRequest;
import software.amazon.awssdk.services.rekognition.model.DeleteProjectResponse;
import software.amazon.awssdk.services.rekognition.model.DescribeProjectsRequest;
import software.amazon.awssdk.services.rekognition.model.DescribeProjectsResponse;
import software.amazon.awssdk.services.rekognition.model.ProjectDescription;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

public class DeleteProject {

    public static final Logger logger =
        Logger.getLogger(DeleteProject.class.getName());

    public static void deleteMyProject(RekognitionClient rekClient, String projectArn)
        throws InterruptedException {

        try {

            logger.log(Level.INFO, "Deleting project: {}", projectArn);

            // Delete the project

            DeleteProjectRequest deleteProjectRequest =
                DeleteProjectRequest.builder().projectArn(projectArn).build();
            DeleteProjectResponse response = rekClient.deleteProject(deleteProjectRequest);

            logger.log(Level.INFO, "Status: {}", response.status());

            // Wait until deletion finishes

            Boolean deleted = false;

            do {

                DescribeProjectsRequest describeProjectsRequest =
                    DescribeProjectsRequest.builder().build();
                DescribeProjectsResponse describeResponse =
                    rekClient.describeProjects(describeProjectsRequest);
                List<ProjectDescription> projectDescriptions =
                    describeResponse.projectDescriptions();

                deleted = true;

            } while (!deleted);

        } catch (RekognitionException e) {
            logger.log(Level.SEVERE, "Error deleting project: " + e.getMessage());
        }
    }
}
```

```
for (ProjectDescription projectDescription : projectDescriptions) {  
    if (Objects.equals(projectDescription.projectArn(), projectArn)) {  
        deleted = false;  
        logger.log(Level.INFO, "Not deleted: {}", projectDescription.projectArn());  
        Thread.sleep(5000);  
        break;  
    }  
}  
} while (Boolean.FALSE.equals(deleted));  
logger.log(Level.INFO, "Project deleted: {} ", projectArn);  
} catch (  
    RekognitionException e) {  
    logger.log(Level.SEVERE, "Client error occurred: {}", e.getMessage());  
    throw e;  
}  
}  
  
public static void main(String args[]) {  
    final String USAGE = "\n" + "Usage: " + "<project_arn>\n\n" + "Where:\n" + "  
    + "    project_arn - The ARN of the project that you want to delete.\n\n";  
    if (args.length != 1) {  
        System.out.println(USAGE);  
        System.exit(1);  
    }  
    String projectArn = args[0];  
    try {  
        RekognitionClient rekClient = RekognitionClient.builder().build();  
  
        // Delete the project  
        deleteMyProject(rekClient, projectArn);  
        System.out.println(String.format("Project deleted: %s", projectArn));  
        rekClient.close();  
    } catch (RekognitionException rekError) {  
        logger.log(Level.SEVERE, "Rekognition client error: {}",  
rekError.getMessage());  
        System.exit(1);  
    }  
    catch (InterruptedException intError) {  
        logger.log(Level.SEVERE, "Exception while sleeping: {}",  
intError.getMessage());  
        System.exit(1);  
    }  
}
```

Describing a project (SDK)

You can use the `DescribeProjects` API to get information about your projects.

To describe a project (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following example code to describe a project. Replace `project_name` with the name of the project that you want to describe. If you don't specify `--project-names`, descriptions for all projects are returned.

AWS CLI

```
aws rekognition describe-projects --project-names "project_name"
```

Python

Use the following code. Supply the following command line parameters:

- `project_name`— the name of the project that you want to describe. If you don't specify a name, descriptions for all projects are returned.

```
# Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
# amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import argparse
import logging
import json
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def display_project_info(project):
    """
    Displays information about a Custom Labels project.
    :param project: The project that you want to display information about.
    """
    print(f"Arn: {project['ProjectArn']}")
    print(f"Status: {project['Status']}")

    if len(project['Datasets']) == 0:
        print("Datasets: None")
    else:
        print("Datasets:")

    for dataset in project['Datasets']:
        print(f"\tCreated: {str(dataset['CreationTimestamp'])}")
        print(f"\tType: {dataset['DatasetType']}")
        print(f"\tARN: {dataset['DatasetArn']}")
        print(f"\tStatus: {dataset['Status']}")
        print(f"\tStatus message: {dataset['StatusMessage']}"
```

```
        print(f"\tStatus code: {dataset['StatusMessageCode']}")
        print()
print()

def describe_projects(rek_client, project_name):
    """
    Describes an Amazon Rekognition Custom Labels project, or all projects.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_name: The project you want to describe. Pass None to describe
    all projects.
    """

    try:
        # Describe the project
        if project_name == None:
            logger.info(f"Describing all projects.")
        else:
            logger.info(f"Describing project: {project_name}.")

        if project_name == None:
            response = rek_client.describe_projects()
        else:
            project_names = json.loads('["' + project_name + '"]')
            response = rek_client.describe_projects(ProjectNames=project_names)

        print('Projects\n-----')
        if len(response['ProjectDescriptions']) == 0:
            print("Project(s) not found.")
        else:
            for project in response['ProjectDescriptions']:
                display_project_info(project)

        logger.info(f"Finished project description.")

    except ClientError as err:
        logger.exception(
            f"Couldn't describe project - {project_name}: {err.response['Error']}"
            ['Message'])
        raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "--project_name", help="The name of the project that you want to
        describe.", required=False
    )

def main():
    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:
        # get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)

        args = parser.parse_args()


```

```
print(f"Describing projects: {args.project_name}")

# Describe the project
rek_client = boto3.client('rekognition')

describe_projects(rek_client,
                  args.project_name)

if args.project_name == None:
    print(f"Finished describing all projects.")
else:
    print(f"Finished describing project {args.project_name}.")

except ClientError as err:
    logger.exception(f"Problem describing project: {err}")
    print(f"Problem describing project: {err}")

if __name__ == "__main__":
    main()
```

Java V2

Use the following code. Supply the following command line parameters:

- `project_name` — the ARN of the project that you want to describe. If you don't specify a name, descriptions for all projects are returned.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import java.net.URI;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DatasetMetadata;
import software.amazon.awssdk.services.rekognition.model.DescribeProjectsRequest;
import software.amazon.awssdk.services.rekognition.model.DescribeProjectsResponse;
import software.amazon.awssdk.services.rekognition.model.ProjectDescription;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

public class DescribeProjects {

    public static final Logger logger =
    Logger.getLogger(DescribeProjects.class.getName());

    public static void describeMyProjects(RekognitionClient rekClient, String
    projectName) {

        DescribeProjectsRequest descProjects = null;

        // If a single project name is supplied, build projectNames argument
        List<String> projectNames = new ArrayList<String>();

        if (projectName == null) {
            descProjects = DescribeProjectsRequest.builder().build();
        } else {
```

```
    projectNames.add(projectName);
    descProjects =
DescribeProjectsRequest.builder().projectNames(projectNames).build();
}

// Display useful information for each project.

DescribeProjectsResponse resp = rekClient.describeProjects(descProjects);

for (ProjectDescription projectDescription : resp.projectDescriptions()) {

    System.out.println("ARN: " + projectDescription.projectArn());
    System.out.println("Status: " + projectDescription.statusAsString());
    if (projectDescription.hasDatasets()) {
        for (DatasetMetadata datasetDescription :
projectDescription.datasets()) {
            System.out.println("\tdataset Type: " +
datasetDescription.datasetTypeAsString());
            System.out.println("\tdataset ARN: " +
datasetDescription.datasetArn());
            System.out.println("\tdataset Status: " +
datasetDescription.statusAsString());
        }
    }
    System.out.println();
}

}

public static void main(String[] args) {

    String projectArn = null;

    // Get command line arguments

    final String USAGE = "\n" + "Usage: " + "<project_name>\n\n" + "Where:\n" +
        "    project_arn - (Optional) The name of the project that you
        want to describe. If not specified, all projects "
        + "are described.\n\n";

    if (args.length > 1) {
        System.out.println(USAGE);
        System.exit(1);
    }

    if (args.length == 1) {
        projectArn = args[0];
    }

    try {

        // Get the Rekognition client
        RekognitionClient rekClient = RekognitionClient.builder().build();

        // Describe projects

        describeMyProjects(rekClient, projectArn);

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    }

}
```

}

Creating a project with AWS CloudFormation

Amazon Rekognition Custom Labels is integrated with AWS CloudFormation, a service that helps you model and set up your AWS resources so that you can spend less time creating and managing your resources and infrastructure. You create a template that describes all the AWS resources that you want, and AWS CloudFormation takes care of provisioning and configuring those resources for you.

You can use AWS CloudFormation to provision and configure Amazon Rekognition Custom Labels projects.

When you use AWS CloudFormation, you can reuse your template to set up your Amazon Rekognition Custom Labels projects consistently and repeatedly. Just describe your projects once, and then provision the same projects over and over in multiple AWS accounts and Regions.

Amazon Rekognition Custom Labels and AWS CloudFormation templates

To provision and configure projects for Amazon Rekognition Custom Labels and related services, you must understand [AWS CloudFormation templates](#). Templates are formatted text files in JSON or YAML. These templates describe the resources that you want to provision in your AWS CloudFormation stacks. If you're unfamiliar with JSON or YAML, you can use AWS CloudFormation Designer to help you get started with AWS CloudFormation templates. For more information, see [What is AWS CloudFormation Designer?](#) in the [AWS CloudFormation User Guide](#).

For reference information about Amazon Rekognition Custom Labels projects, including examples of JSON and YAML templates, see [Rekognition resource type reference](#).

Learn more about AWS CloudFormation

To learn more about AWS CloudFormation, see the following resources:

- [AWS CloudFormation](#)
- [AWS CloudFormation User Guide](#)
- [AWS CloudFormation API Reference](#)
- [AWS CloudFormation Command Line Interface User Guide](#)

Managing datasets

A dataset contains the images and assigned labels that you use to train or test a model. The topics in this section show you how to manage a dataset with the Amazon Rekognition Custom Labels console and the AWS SDK.

Topics

- [Adding a dataset to a project \(p. 212\)](#)
- [Adding more images to a dataset \(p. 218\)](#)
- [Describing a dataset \(SDK\) \(p. 224\)](#)
- [Listing dataset entries \(SDK\) \(p. 227\)](#)
- [Distributing a training dataset \(SDK\) \(p. 231\)](#)

- [Deleting a dataset \(p. 238\)](#)
- [Creating a manifest file \(p. 242\)](#)

Adding a dataset to a project

You can add a training dataset or a test dataset to an existing project. If you want to replace an existing dataset, first delete the existing dataset. For more information, see [Deleting a dataset \(p. 238\)](#). Then, add the new dataset.

Topics

- [Adding a dataset to a project \(Console\) \(p. 212\)](#)
- [Adding a dataset to a project \(SDK\) \(p. 212\)](#)

Adding a dataset to a project (Console)

You can add a training or test dataset to a project by using the Amazon Rekognition Custom Labels console.

To add a dataset to a project

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. In the left pane, choose **Use Custom Labels**. The Amazon Rekognition Custom Labels landing page is shown.
3. In the left navigation pane, choose **Projects**. The Projects view is shown.
4. Choose the project to which you want to add a dataset.
5. In the left navigation pane, under the project name, choose **Datasets**.
6. If the project doesn't have an existing dataset, the **Create dataset** page is shown. Do the following:
 - a. On the **Create dataset** page, enter the image source information. For more information, see [the section called "Creating datasets \(Console\)" \(p. 59\)](#).
 - b. Choose **Create dataset** to create the dataset.
7. If the project has an existing dataset (training or test), the project details page is shown. Do the following:
 - a. On the project details page, choose **Actions**.
 - b. If you want to add a training dataset, choose **Create training dataset**.
 - c. If you want to add a test dataset, choose **Create test dataset**.
 - d. On the **Create dataset** page, enter the image source information. For more information, see [the section called "Creating datasets \(Console\)" \(p. 59\)](#).
 - e. Choose **Create dataset** to create the dataset.
8. Add images to your dataset. For more information, see [Adding more images \(console\) \(p. 218\)](#).
9. Add labels to your dataset. For more information, see [Add new labels \(Console\) \(p. 100\)](#).
10. Add labels to your images. If you're adding image-level labels, see [the section called "Assigning image-level labels to an image" \(p. 102\)](#). If you're adding bounding boxes, see [Labeling objects with bounding boxes \(p. 104\)](#). For more information, see [Purposing datasets \(p. 55\)](#).

Adding a dataset to a project (SDK)

You can add a train or test dataset to an existing project in the following ways:

- Create a dataset using a manifest file. For more information, see [Creating a dataset with a manifest file \(SDK\) \(p. 66\)](#).
- Create an empty dataset and populate the dataset afterwards. The following example shows how to create an empty dataset. To add entries after you create an empty dataset, see [Adding more images to a dataset \(p. 218\)](#).

To add a dataset to a project (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with AmazonRekognitionFullAccess permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following examples to add JSON lines to a dataset.

CLI

Replace `project_arn` with the project that you want to add the dataset set to. Replace `dataset_type` with `TRAIN` to create a training dataset, or `TEST` to create a test dataset. to escape any special characters within the JSON Line.

```
aws rekognition create-dataset --project-arn "project_arn" \
--dataset-type dataset_type
```

Python

Use the following code to create a dataset. Supply the following command line options:

- `project_arn` — the ARN of the project that you want to add the test dataset to.
- `type` — the type of dataset that you want to create (train or test)

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import argparse
import logging
import time
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def create_empty_dataset(rek_client, project_arn, dataset_type):
    """
    Creates an empty Amazon Rekognition Custom Labels dataset.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_arn: The ARN of the project in which you want to create a dataset.
    :param dataset_type: The type of the dataset that you want to create (train or test).
    """

    try:
        #Create the dataset
        logger.info(f"Creating empty {dataset_type} dataset for project {project_arn}")
    except ClientError as error:
        logger.error(f"Error creating dataset: {error}")
        raise
```

```
dataset_type=dataset_type.upper()

response = rek_client.create_dataset(
    ProjectArn=project_arn, DatasetType=dataset_type
)

dataset_arn=response['DatasetArn']

logger.info(f"dataset ARN: {dataset_arn}")

finished=False
while finished==False:

    dataset=rek_client.describe_dataset(DatasetArn=dataset_arn)

    status=dataset['DatasetDescription']['Status']

    if status == "CREATE_IN_PROGRESS":

        logger.info((f"Creating dataset: {dataset_arn} "))
        time.sleep(5)
        continue

    if status == "CREATE_COMPLETE":
        logger.info(f"Dataset created: {dataset_arn}")
        finished=True
        continue

    if status == "CREATE_FAILED":
        logger.exception(f"Dataset creation failed: {status} : {dataset_arn}")
        raise Exception (f"Dataset creation failed: {status} : {dataset_arn}")

        logger.exception(f"Failed. Unexpected state for dataset creation: {status} : {dataset_arn}")
        raise Exception(f"Failed. Unexpected state for dataset creation: {status} : {dataset_arn}")

    return dataset_arn

except ClientError as err:
    logger.exception(f"Couldn't create dataset: {err.response['Error']['Message']}")
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project in which you want to create the empty dataset."
    )

    parser.add_argument(
        "dataset_type", help="The type of the empty dataset that you want to create (train or test)."
    )

def main():
```

```
logging.basicConfig(level=logging.INFO, format"%(levelname)s: %(message)s")  
  
try:  
  
    #get command line arguments  
    parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)  
    add_arguments(parser)  
    args = parser.parse_args()  
  
    print(f"Creating empty {args.dataset_type} dataset for project  
{args.project_arn}")  
  
    #Create the empty dataset  
    rek_client=boto3.client('rekognition')  
  
    dataset_arn=create_empty_dataset(rek_client,  
                                    args.project_arn,  
                                    args.dataset_type.lower())  
  
    print(f"Finished creating empty dataset: {dataset_arn}")  
  
except ClientError as err:  
    logger.exception(f"Problem creating empty dataset: {err}")  
    print(f"Problem creating empty dataset: {err}")  
except Exception as err:  
    logger.exception(f"Problem creating empty dataset: {err}")  
    print(f"Problem creating empty dataset: {err}")  
  
if __name__ == "__main__":  
    main()
```

Java 2

Use the following code to create a dataset. Supply the following command line options:

- `project_arn` — the ARN of the project that you want to add the test dataset to.
- `type` — the type of dataset that you want to create (train or test)

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.  
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/  
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)  
  
import software.amazon.awssdk.services.rekognition.RekognitionClient;  
import software.amazon.awssdk.services.rekognition.model.CreateDatasetRequest;  
import software.amazon.awssdk.services.rekognition.model.CreateDatasetResponse;  
import software.amazon.awssdk.services.rekognition.model.DatasetDescription;  
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;  
import software.amazon.awssdk.services.rekognition.model.DatasetType;  
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;  
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;  
import software.amazon.awssdk.services.rekognition.model.RekognitionException;  
  
import java.net.URI;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
public class CreateEmptyDataset {
```

```
public static final Logger logger =
Logger.getLogger(CreateEmptyDataset.class.getName());

public static String createMyEmptyDataset(RekognitionClient rekClient, String
projectArn, String datasetType)
throws Exception, RekognitionException {

try {

logger.log(Level.INFO, "Creating empty {0} dataset for project : {1}",
new Object[] { datasetType.toString(), projectArn });

DatasetType requestDatasetType = null;

switch (datasetType) {
case "train":
requestDatasetType = DatasetType.TRAIN;
break;
case "test":
requestDatasetType = DatasetType.TEST;
break;
default:
logger.log(Level.SEVERE, "Unrecognized dataset type: {0}",
datasetType);
throw new Exception("Unrecognized dataset type: " + datasetType);
}

CreateDatasetRequest createDatasetRequest =
CreateDatasetRequest.builder().projectArn(projectArn)
.datasetType(requestDatasetType).build();

CreateDatasetResponse response =
rekClient.createDataset(createDatasetRequest);

boolean created = false;

//Wait until updates finishes

do {

DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder()
.datasetArn(response.datasetArn()).build();
DescribeDatasetResponse describeDatasetResponse =
rekClient.describeDataset(describeDatasetRequest);

DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

DatasetStatus status = datasetDescription.status();

logger.log(Level.INFO, "Creating dataset ARN: {0} ",
response.datasetArn());

switch (status) {

case CREATE_COMPLETE:
logger.log(Level.INFO, "Dataset created");
created = true;
break;

case CREATE_IN_PROGRESS:
Thread.sleep(5000);
break;
}
}
}
}
```

```
        case CREATE_FAILED:
            String error = "Dataset creation failed: " +
datasetDescription.statusAsString() + " "
                           + datasetDescription.statusMessage() + " " +
response.datasetArn();
            logger.log(Level.SEVERE, error);
            throw new Exception(error);

        default:
            String unexpectedError = "Unexpected creation state: " +
datasetDescription.statusAsString() + " "
                           + datasetDescription.statusMessage() + " " +
response.datasetArn();
            logger.log(Level.SEVERE, unexpectedError);
            throw new Exception(unexpectedError);
    }

} while (created == false);

return response.datasetArn();

} catch (RekognitionException e) {
    logger.log(Level.SEVERE, "Could not create dataset: {0}",
e.getMessage());
    throw e;
}

}

public static void main(String args[]) {

    String datasetType = null;
    String datasetArn = null;
    String projectArn = null;

    final String USAGE = "\n" + "Usage: " + "<project_arn> <dataset_type>\n\n" +
"Where:\n" +
        "    + " project_arn - the ARN of the project that you want to add
copy the dataset to.\n\n" +
        "    + " dataset_type - the type of the empty dataset that you want to
create (train or test).\n\n";

    if (args.length != 2) {
        System.out.println(USAGE);
        System.exit(1);
    }

    projectArn = args[0];
    datasetType = args[1];

    try {

        // Get the Rekognition client
        RekognitionClient rekClient = RekognitionClient.builder().build();

        // Create the dataset
        datasetArn = createMyEmptyDataset(rekClient, projectArn, datasetType);

        System.out.println(String.format("Created dataset: %s", datasetArn));

        rekClient.close();

    } catch (RekognitionException rekError) {
```

```
        logger.log(Level.SEVERE, "Rekognition client error: {0}",  
        rekError.getMessage());  
        System.exit(1);  
    } catch (Exception rekError) {  
        logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());  
        System.exit(1);  
    }  
}  
}
```

3. Add images to the dataset. For more information, see [Adding more images \(SDK\) \(p. 218\)](#).

Adding more images to a dataset

You can add more images to your datasets by using the Amazon Rekognition Custom Labels console or by calling the `UpdateDatasetEntries` API.

Adding more images (console)

When you use the Amazon Rekognition Custom Labels console, you upload images from your local computer. The images are added to the Amazon S3 bucket location (console or external) where the images used to create the dataset are stored.

To add more images to your dataset (console)

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. In the left pane, choose **Custom Labels**. The Amazon Rekognition Custom Labels landing page is shown.
3. In the left navigation pane, choose **Projects**. The Projects view is shown.
4. Choose the project that you want to use.
5. In the left navigation pane, under the project name, choose **Dataset**.
6. Choose **Actions** and select the dataset that you want to add images to.
7. Choose the images you want to upload to the dataset. You can drag the images or choose the images that you want to upload from your local computer. You can upload up to 30 images at a time.
8. Choose **Upload images**.
9. Choose **Save changes**.
10. Label the images. For more information, see [Labeling images \(p. 99\)](#).

Adding more images (SDK)

`UpdateDatasetEntries` updates or adds JSON lines to a manifest file. You pass the JSON lines as a `byte64` encoded data object in the `GroundTruth` field. If you are using an AWS SDK to call `UpdateDatasetEntries`, the SDK encodes the data for you. Each JSON line contains information for a single image, such as assigned labels or bounding box information. For example:

```
{"source-ref":"s3://bucket/image","BB":{"annotations":  
[{"left":1849,"top":1039,"width":422,"height":283,"class_id":0},  
 {"left":1849,"top":1340,"width":443,"height":415,"class_id":1},  
 {"left":2637,"top":1380,"width":676,"height":338,"class_id":2},  
 {"left":2634,"top":1051,"width":673,"height":338,"class_id":3}], "image_size":  
 [{"width":4000,"height":2667,"depth":3}]}}, "BB-metadata":{"job-name":"labeling-job/  
BB","class-map":
```

```
{"0":"comparator","1":"pot_resistor","2":"ir_phototransistor","3":"ir_led"}, "human-annotated": "yes", "objects": [{"confidence":1}, {"confidence":1}, {"confidence":1}, {"confidence":1}], "creation-date": "2021-06-22T10:11:18.006Z", "type": "groundtruth/object-detection"}
```

For more information, see [Creating a manifest file \(p. 242\)](#).

Use source-ref field as a key to identify images that you want to update. If the dataset doesn't contain a matching source-ref field value, the JSON line is added as a new image.

To add more images to a dataset (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with AmazonRekognitionFullAccess permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following examples to add JSON lines to a dataset.

CLI

Replace the value of GroundTruth with the JSON Lines that you want to use. You need to escape any special characters within the JSON Line.

```
aws rekognition update-dataset-entries \
  --dataset-arn dataset_arn \
  --changes '{"GroundTruth": "{\"source-ref\": \"s3://your_bucket/your_image \
  \", \"BB\": {\"annotations\": [{\"left\": 1776, \"top\": 1017, \"width\": 458, \"height \
  \": 317, \"class_id\": 0}, {\"left\": 1797, \"top\": 1334, \"width\": 418, \"height \
  \": 415, \"class_id\": 1}, {\"left\": 2597, \"top\": 1361, \"width\": 655, \"height \
  \": 329, \"class_id\": 2}, {\"left\": 2581, \"top\": 1020, \"width\": 689, \"height \
  \": 338, \"class_id\": 3}], \"image_size\": [{\"width\": 4000, \"height\": 2667, \"depth \
  \": 3}], \"BB-metadata\": {\"job-name\": \"labeling-job/BB\", \"class-map\": {\"0\": \"comparator\", \
  \"1\": \"pot_resistor\", \"2\": \"ir_phototransistor\", \"3\": \"ir_led\"}, \"human-annotated \
  \": \"yes\", \"objects\": [{\"confidence\": 1}, {\"confidence\": 1}, {\"confidence\": 1}, \
  {\"confidence\": 1}], \"creation-date\": \"2021-06-22T10:10:48.492Z\", \"type\": \
  \"groundtruth/object-detection\"} }' \
  --cli-binary-format raw-in-base64-out
```

Python

Use the following code. Supply the following command line parameters:

- dataset_arn—the ARN of the dataset that you want to update.
- updates_file—the file that contains the JSON Line updates.

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import argparse
import logging
import time
import json

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)
```

```
def update_dataset_entries(rek_client, dataset_arn, updates_file):
    """
    Adds dataset entries to an Amazon Rekognition Custom Labels dataset.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param dataset_arn: The ARN of the dataset that you want to update.
    :param updates_file: The manifest file of JSON Lines that contains the
    updates.
    """

    try:
        status=""
        status_message=""

        #Update dataset entries
        logger.info(f"Updating dataset {dataset_arn}")

        with open(updates_file) as f:
            manifest_file = f.read()

            changes=json.loads('{"GroundTruth" : ' +
                               json.dumps(manifest_file) +
                               '}')

            rek_client.update_dataset_entries(
                Changes=changes, DatasetArn=dataset_arn
            )

        finished=False
        while finished==False:

            dataset=rek_client.describe_dataset(DatasetArn=dataset_arn)

            status=dataset['DatasetDescription']['Status']
            status_message=dataset['DatasetDescription']['StatusMessage']

            if status == "UPDATE_IN_PROGRESS":

                logger.info((f"Updating dataset: {dataset_arn} "))
                time.sleep(5)
                continue

            if status == "UPDATE_COMPLETE":
                logger.info(f"Dataset updated: {status} : {status_message} : {dataset_arn}")
                finished=True
                continue

            if status == "UPDATE_FAILED":
                logger.exception(f"Dataset update failed: {status} : {status_message} : {dataset_arn}")
                raise Exception(f"Dataset update failed: {status} : {status_message} : {dataset_arn}")

            logger.exception(f"Failed. Unexpected state for dataset update: {status} : {status_message} : {dataset_arn}")
            raise Exception(f"Failed. Unexpected state for dataset update: {status} : {status_message} : {dataset_arn}")

        logger.info(f"Added entries to dataset")

    return status, status_message
```

```
        except ClientError as err:
            logger.exception(f"Couldn't update dataset: {err.response['Error']}")
            raise

    def add_arguments(parser):
        """
        Adds command line arguments to the parser.
        :param parser: The command line parser.
        """

        parser.add_argument(
            "dataset_arn", help="The ARN of the dataset that you want to update."
        )

        parser.add_argument(
            "updates_file", help="The manifest file of JSON Lines that contains the
            updates."
        )

    def main():

        logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

        try:

            #get command line arguments
            parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
            add_arguments(parser)
            args = parser.parse_args()

            print(f"Updating dataset {args.dataset_arn} with entries from
            {args.updates_file}.")

            #Update the dataset
            rek_client=boto3.client('rekognition')

            status, status_message=update_dataset_entries(rek_client,
                args.dataset_arn,
                args.updates_file)

            print(f"Finished updates dataset: {status} : {status_message}")

        except ClientError as err:
            logger.exception(f"Problem updating dataset: {err}")
            print(f"Problem updating dataset: {err}")
        except Exception as err:
            logger.exception(f"Problem updating dataset: {err}")
            print(f"Problem updating dataset: {err}")

    if __name__ == "__main__":
        main()
```

Java 2

- `dataset_arn`— the ARN of the dataset that you want to update.
- `update_file`— the file that contains the JSON Line updates.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/  
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)  
  
import software.amazon.awssdk.core.SdkBytes;  
import software.amazon.awssdk.services.rekognition.RekognitionClient;  
import software.amazon.awssdk.services.rekognition.model.DatasetChanges;  
import software.amazon.awssdk.services.rekognition.model.DatasetDescription;  
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;  
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;  
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;  
import software.amazon.awssdk.services.rekognition.model.RekognitionException;  
import  
    software.amazon.awssdk.services.rekognition.model.UpdateDatasetEntriesRequest;  
import  
    software.amazon.awssdk.services.rekognition.model.UpdateDatasetEntriesResponse;  
  
import java.io.FileInputStream;  
import java.io.InputStream;  
import java.net.URI;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
public class UpdateDatasetEntries {  
  
    public static final Logger logger =  
Logger.getLogger(UpdateDatasetEntries.class.getName());  
  
    public static String updateMyDataset(RekognitionClient rekClient, String  
datasetArn,  
        String updateFile  
    ) throws Exception, RekognitionException {  
  
        try {  
  
            logger.log(Level.INFO, "Updating dataset {0}",  
                new Object[] { datasetArn});  
  
            InputStream sourceStream = new FileInputStream(updateFile);  
            SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);  
  
            DatasetChanges datasetChanges = DatasetChanges.builder()  
                .groundTruth(sourceBytes).build();  
  
            UpdateDatasetEntriesRequest updateDatasetEntriesRequest =  
UpdateDatasetEntriesRequest.builder()  
                .changes(datasetChanges)  
                .datasetArn(datasetArn)  
                .build();  
  
            UpdateDatasetEntriesResponse response =  
rekClient.updateDatasetEntries(updateDatasetEntriesRequest);  
  
            boolean updated = false;  
  
            //Wait until update completes  
  
            do {  
  
                DescribeDatasetRequest describeDatasetRequest =  
DescribeDatasetRequest.builder()  
                    .datasetArn(datasetArn).build();  
                DescribeDatasetResponse describeDatasetResponse =  
rekClient.describeDataset(describeDatasetRequest);
```

```
DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

DatasetStatus status = datasetDescription.status();

logger.log(Level.INFO, " dataset ARN: {0} ", datasetArn);

switch (status) {

    case UPDATE_COMPLETE:
        logger.log(Level.INFO, "Dataset updated");
        updated = true;
        break;

    case UPDATE_IN_PROGRESS:
        Thread.sleep(5000);
        break;

    case UPDATE_FAILED:
        String error = "Dataset update failed: " +
datasetDescription.statusAsString() + " "
+ datasetDescription.statusMessage() + " " +
datasetArn;
        logger.log(Level.SEVERE, error);
        throw new Exception(error);

    default:
        String unexpectedError = "Unexpected update state: " +
datasetDescription.statusAsString() + " "
+ datasetDescription.statusMessage() + " " +
datasetArn;
        logger.log(Level.SEVERE, unexpectedError);
        throw new Exception(unexpectedError);
    }

} while (updated == false);

return datasetArn;

} catch (RekognitionException e) {
    logger.log(Level.SEVERE, "Could not update dataset: {0}",
e.getMessage());
    throw e;
}

}

public static void main(String args[]) {

    String updatesFile = null;
    String datasetArn = null;

    final String USAGE = "\n" + "Usage: " + "<project_arn> <dataset_arn>
<updates_file>\n\n" + "Where:\n"
+ "    dataset_arn - the ARN of the dataset that you want to update.
\n\n"
+ "    update_file - The file that includes in JSON Line updates.\n
\n";
    if (args.length != 2) {
        System.out.println(USAGE);
        System.exit(1);
    }

    datasetArn = args[0];
```

```
updatesFile = args[1];

try {

    // Get the Rekognition client
    RekognitionClient rekClient = RekognitionClient.builder().build();

    // Update the dataset
    datasetArn = updateMyDataset(rekClient, datasetArn, updatesFile);

    System.out.println(String.format("Dataset updated: %s", datasetArn));

    rekClient.close();

} catch (RekognitionException rekError) {
    logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
    System.exit(1);
} catch (Exception rekError) {
    logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());
    System.exit(1);
}

}

}
```

Describing a dataset (SDK)

You can use the `DescribeDataset` API to get information about a dataset.

To describe a dataset (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following example code to describe a dataset.

AWS CLI

Change the value of `dataset-arn` to the ARN of the dataset that you want to describe.

```
aws rekognition describe-dataset --dataset-arn dataset_arn
```

Python

Use the following code. Supply the following command line parameters:

- `dataset_arn` — the ARN of the dataset that you want to describe.

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-
rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)
```

```
import boto3
import argparse
import logging

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def describe_dataset(rek_client, dataset_arn):
    """
    Describes an Amazon Rekognition Custom Labels dataset.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param dataset_arn: The ARN of the dataset that you want to describe.

    """

    try:
        #Describe the dataset
        logger.info(f"Describing dataset {dataset_arn}")

        dataset=rek_client.describe_dataset(DatasetArn=dataset_arn)

        description = dataset['DatasetDescription']

        print(f"Created: {str(description['CreationTimestamp'])}")
        print(f"Updated: {str(description['LastUpdatedTimestamp'])}")
        print(f"Status: {description['Status']}")
        print(f"Status message: {description['StatusMessage']}")
        print(f"Status code: {description['StatusMessageCode']}")
        print("Stats:")
        print(f"\tLabeled entries: {description['DatasetStats']['LabeledEntries']}")
        print(f"\tTotal entries: {description['DatasetStats']['TotalEntries']}")
        print(f"\tTotal labels: {description['DatasetStats']['TotalLabels']}")

    except ClientError as err:
        logger.exception(f"Couldn't describe dataset: {err.response['Error']['Message']}")
        raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "dataset_arn", help="The ARN of the dataset that you want to describe."
    )

def main():
    logging.basicConfig(level=logging.INFO, format"%(levelname)s: %(message)s")

    try:
        #get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Describing dataset {args.dataset_arn}")

    
```

```
#Describe the dataset
rek_client=boto3.client('rekognition')

    describe_dataset(rek_client,
                      args.dataset_arn
    )

    print(f"Finished describing dataset: {args.dataset_arn}")

except ClientError as err:
    logger.exception(f"Problem describing dataset: {err}")
    print(f"Problem describing dataset: {err}")
except Exception as err:
    logger.exception(f"Problem describing dataset: {err}")
    print(f"Problem describing dataset: {err}")

if __name__ == "__main__":
    main()
```

Java V2

- `dataset_arn` — the ARN of the dataset that you want to describe.

```
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DatasetDescription;
import software.amazon.awssdk.services.rekognition.model.DatasetStats;
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;

public class DescribeDataset {

    public static final Logger logger =
    Logger.getLogger(DescribeDataset.class.getName());

    public static void describeMyDataset(RekognitionClient rekClient, String
datasetArn) {

        try {

            DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder().datasetArn(datasetArn)
                .build();
            DescribeDatasetResponse describeDatasetResponse =
rekClient.describeDataset(describeDatasetRequest);

            DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();
            DatasetStats datasetStats = datasetDescription.datasetStats();

            System.out.println("ARN: " + datasetArn);
            System.out.println("Created: " +
datasetDescription.creationTimestamp().toString());
        }
    }
}
```

```
        System.out.println("Updated: " +
datasetDescription.lastUpdatedTimestamp().toString());
        System.out.println("Status: " + datasetDescription.statusAsString());
        System.out.println("Message: " + datasetDescription.statusMessage());
        System.out.println("Total Labels: " +
datasetStats.totalLabels().toString());
        System.out.println("Total entries: " +
datasetStats.totalEntries().toString());
        System.out.println("Entries with labels: " +
datasetStats.labeledEntries().toString());
        System.out.println("Entries with at least 1 error: " +
datasetStats.errorEntries().toString());

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        throw rekError;
    }
}

public static void main(String args[]) {
    final String USAGE = "\n" + "Usage: " + "<dataset_arn>\n\n" + "Where:\n" +
        "    dataset_arn - The ARN of the dataset that you want to
describe.\n\n";
    if (args.length != 1) {
        System.out.println(USAGE);
        System.exit(1);
    }
    String datasetArn = args[0];
    try {
        // Get the Rekognition client
        RekognitionClient rekClient = RekognitionClient.builder().build();

        // Describe the dataset
        describeMyDataset(rekClient, datasetArn);

        rekClient.close();
    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    }
}
}
```

Listing dataset entries (SDK)

You can use the `ListDatasetEntries` API to list the JSON lines for each image in a dataset. For more information, see [Creating a manifest file \(p. 242\)](#).

To list dataset entries (SDK)

1. If you haven't already:

- a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following example code list the entries in a dataset

AWS CLI

Change the value of `dataset_arn` to the ARN of the dataset that you want to list.

```
aws rekognition list-dataset-entries --dataset-arn dataset_arn
```

To list only JSON lines with errors, specify `has-errors`.

```
aws rekognition list-dataset-entries --dataset-arn dataset_arn \  
--has-errors
```

Python

Use the following code. Supply the following command line parameters:

- `dataset_arn` — the ARN of the dataset that you want to list.
- `show_errors_only` — specify `true` if you want to see errors only. `false` otherwise.

```
# Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/  
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)  
  
import boto3  
import argparse  
import logging  
  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)  
  
def list_dataset_entries(rek_client, dataset_arn, show_errors):  
    """  
    Lists the entries in an Amazon Rekognition Custom Labels dataset.  
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.  
    :param dataset_arn: The ARN of the dataet that you want to use.  
    """  
  
    try:  
        # List the entries  
        logger.info(f"Listing dataset entries for the dataset {dataset_arn}.")  
  
        finished = False  
        count=0  
        next_token = ""  
        show_errors_only = False  
  
        if show_errors.lower() == "true":  
            show_errors_only=True
```

```
while finished == False:

    response = rek_client.list_dataset_entries(
        DatasetArn=dataset_arn,
        HasErrors=show_errors_only,
        MaxResults=100,
        NextToken=next_token)

    count += len(response['DatasetEntries'])

    for entry in response['DatasetEntries']:
        print(entry)

        if not 'NextToken' in response:
            finished = True
            logger.info(f"No more entries. Total:{count}")
        else:
            next_token=next_token=response['NextToken']
            logger.info(f"Getting more entries. Total so far :{count}")

except ClientError as err:
    logger.exception(
        f"Couldn't list dataset: {err.response['Error']['Message']}")
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "dataset_arn", help="The ARN of the dataset that you want to list."
    )

    parser.add_argument(
        "show_errors_only", help="true if you want to see errors only. false otherwise."
    )

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Listing entries for dataset {args.dataset_arn}")

        # List the dataset entries
        rek_client=boto3.client('rekognition')

        list_dataset_entries(rek_client,
                            args.dataset_arn,
                            args.show_errors_only)
```

```
        print(f"Finished listing entries for dataset: {args.dataset_arn}")

    except ClientError as err:
        logger.exception(f"Problem listing dataset: {err}")
        print(f"Problem listing dataset: {err}")
    except Exception as err:
        logger.exception(f"Problem listing dataset: {err}")
        print(f"Problem listing dataset: {err}")

if __name__ == "__main__":
    main()
```

Java 2

Use the following code. Supply the following command line parameters:

- `dataset_arn` — the ARN of the dataset that you want to list.
- `show_errors_only` — specify `true` if you want to see errors only. `false` otherwise.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.ListDatasetEntriesRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;
import
software.amazon.awssdk.services.rekognition.paginators.ListDatasetEntriesIterable;

import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;

public class ListDatasetEntries {

    public static final Logger logger =
Logger.getLogger(ListDatasetEntries.class.getName());

    public static void listMyDatasetEntries(RekognitionClient rekClient, String
datasetArn, boolean showErrorsOnly)
        throws Exception, RekognitionException {

        try {

            logger.log(Level.INFO, "Listing dataset {0}", new Object[]
{ datasetArn });

            ListDatasetEntriesRequest listDatasetEntriesRequest =
ListDatasetEntriesRequest.builder()

.hasErrors(showErrorsOnly).datasetArn(datasetArn).maxResults(1).build();

            ListDatasetEntriesIterable datasetEntriesList = rekClient
.listDatasetEntriesPaginator(listDatasetEntriesRequest);

            datasetEntriesList.stream().flatMap(r -> r.datasetEntries().stream()
.forEach(datasetEntry ->
System.out.println(datasetEntry.toString())));
        } catch (RekognitionException e) {
```

```
        logger.log(Level.SEVERE, "Could not update dataset: {0}",  
        e.getMessage());  
        throw e;  
    }  
  
}  
  
public static void main(String args[]) {  
  
    boolean showErrorsOnly = false;  
    String datasetArn = null;  
  
    final String USAGE = "\n" + "Usage: " + "<project_arn> <dataset_arn>  
<updates_file>\n\n" + "Where:\n" + "  
        + " dataset_arn - the ARN of the dataset that you want to update.  
\n\n" + " show_errors_only - true to show only errors. false otherwise.  
\n\n";  
  
    if (args.length != 2) {  
        System.out.println(USAGE);  
        System.exit(1);  
    }  
  
    datasetArn = args[0];  
    if (args[1].toLowerCase().equals("true")) {  
  
        showErrorsOnly = true;  
    }  
  
    try {  
  
        // Get the Rekognition client  
        RekognitionClient rekClient = RekognitionClient.builder().build();  
  
        // list the dataset  
  
        listMyDatasetEntries(rekClient, datasetArn, showErrorsOnly);  
  
        System.out.println(String.format("Finished listing entries for : %s",  
datasetArn));  
  
        rekClient.close();  
  
    } catch (RekognitionException rekError) {  
        logger.log(Level.SEVERE, "Rekognition client error: {0}",  
rekError.getMessage());  
        System.exit(1);  
    } catch (Exception rekError) {  
        logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());  
        System.exit(1);  
    }  
}  
}
```

Distributing a training dataset (SDK)

Amazon Rekognition Custom Labels requires a training dataset and a test dataset to train your model.

If you are using the API, you can use the [DistributeDatasetEntries](#) API to distribute 20% of the training dataset into an empty test dataset. Distributing the training dataset can be useful if you only have a

single manifest file available. Use the single manifest file to create your training dataset. Then create an empty test dataset and use `DistributeDatasetEntries` to populate the test dataset.

Note

If you are using the Amazon Rekognition Custom Labels console and start with a single dataset project, Amazon Rekognition Custom Labels splits (distributes) the training dataset, during training, to create a test dataset. 20% of the training dataset entries are moved to the test dataset.

To distribute a training dataset (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Create a project. For more information, see [Creating an Amazon Rekognition Custom Labels project \(SDK\) \(p. 51\)](#).
3. Create your training dataset. For information about datasets, see [Creating training and test datasets \(p. 54\)](#).
4. Create an empty test dataset.
5. Use the following example code to distribute 20% of the training dataset entries into the test dataset. You can get the Amazon Resource Names (ARN) for a project's datasets by calling `DescribeProjects`. For example code, see [Describing a project \(SDK\) \(p. 207\)](#).

AWS CLI

Change the value of `training_dataset_arn` and `test_dataset_arn` with the ARNs of the datasets that you want to use.

```
aws rekognition distribute-dataset-entries --datasets ['{"Arn": "training_dataset_arn"}, {"Arn": "test_dataset_arn"}']
```

Python

Use the following code. Supply the following command line parameters:

- `training_dataset_arn` — the ARN of the training dataset that you distribute entries from.
- `test_dataset_arn` — the ARN of the test dataset that you distribute entries to.

```
# Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
# amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import argparse
import logging
import time
import json
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def check_dataset_status(rek_client, dataset_arn):
    finished = False
```

```
status = ""
status_message = ""

while finished == False:

    dataset = rek_client.describe_dataset(DatasetArn=dataset_arn)

    status = dataset['DatasetDescription']['Status']
    status_message = dataset['DatasetDescription']['StatusMessage']

    if status == "UPDATE_IN_PROGRESS":

        logger.info((f"Distributing dataset: {dataset_arn} "))
        time.sleep(5)
        continue

    if status == "UPDATE_COMPLETE":
        logger.info(
            f"Dataset distribution complete: {status} : {status_message} :
{dataset_arn}")
        finished = True
        continue

    if status == "UPDATE_FAILED":
        logger.exception(
            f"Dataset distribution failed: {status} : {status_message} :
{dataset_arn}")
        finished = True
        break

    logger.exception(
        f"Failed. Unexpected state for dataset distribution: {status} :
{status_message} : {dataset_arn}")
    finished = True
    status_message = "An unexpected error occurred while distributing the
dataset"
    break

return status, status_message

def distribute_dataset_entries(rek_client, training_dataset_arn, test_dataset_arn):
    """
    Distributes 20% of the supplied training dataset into the supplied test
    dataset.

    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param training_dataset_arn: The ARN of the training dataset that you
    distribute entries from.
    :param test_dataset_arn: The ARN of the test dataset that you distribute
    entries to.
    """

    try:
        # List dataset labels
        logger.info(f"Distributing training dataset entries
({training_dataset_arn}) into test dataset ({test_dataset_arn})."
)

        datasets = json.loads(
            '[{"Arn" : "' + str(training_dataset_arn) + '"}, {"Arn" : "' +
            str(test_dataset_arn) + '"}]')

        rek_client.distribute_dataset_entries(
            Datasets=datasets
        )
    
```

```
    training_dataset_status, training_dataset_status_message =
check_dataset_status(
    rek_client, training_dataset_arn)
    test_dataset_status, test_dataset_status_message = check_dataset_status(
    rek_client, test_dataset_arn)

    if training_dataset_status == 'UPDATE_COMPLETE' and test_dataset_status ==
"UPDATE_COMPLETE":
        print(f"Distribution complete")
    else:
        print("Distribution failed:")
        print(
            f"\ttraining dataset: {training_dataset_status} :
{training_dataset_status_message}")
        print(
            f"\ttest dataset: {test_dataset_status} :
{test_dataset_status_message}")

except ClientError as err:
    logger.exception(
        f"Couldn't distribute dataset: {err.response['Error']['Message']}")
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "training_dataset_arn", help="The ARN of the training dataset that you want
        to distribute from."
    )

    parser.add_argument(
        "test_dataset_arn", help="The ARN of the test dataset that you want to
        distribute to."
    )

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(
            f"Distributing training dataset entries ({args.training_dataset_arn})
            into test dataset ({args.test_dataset_arn}).")

        # Distribute the datasets

        rek_client = boto3.client('rekognition')

        distribute_dataset_entries(rek_client,
                                    args.training_dataset_arn,
                                    args.test_dataset_arn)

        print(f"Finished distributing datasets.")

    
```

```
        except ClientError as err:
            logger.exception(f"Problem distributing datasets: {err}")
            print(f"Problem listing dataset labels: {err}")
        except Exception as err:
            logger.exception(f"Problem distributing datasets: {err}")
            print(f"Problem distributing datasets: {err}")

if __name__ == "__main__":
    main()
```

Java 2

Use the following code. Supply the following command line parameters:

- `training_dataset_arn` — the ARN of the training dataset that you distribute entries from.
- `test_dataset_arn` — the ARN of the test dataset that you distribute entries to.

```
//Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DatasetDescription;
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.DistributeDataset;
import
software.amazon.awssdk.services.rekognition.model.DistributeDatasetEntriesRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;

public class DistributeDatasetEntries {

    public static final Logger logger =
Logger.getLogger(DistributeDatasetEntries.class.getName());

    public static DatasetStatus checkDatasetStatus(RekognitionClient rekClient,
String datasetArn)
        throws Exception, RekognitionException {

        boolean distributed = false;
        DatasetStatus status = null;

        // Wait until distribution completes

        do {

            DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder().datasetArn(datasetArn)
                .build();
            DescribeDatasetResponse describeDatasetResponse =
rekClient.describeDataset(describeDatasetRequest);

            DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

            status = datasetDescription.status();

        } while (!distributed);
    }
}
```

```
        logger.log(Level.INFO, " dataset ARN: {0} ", datasetArn);

        switch (status) {

            case UPDATE_COMPLETE:
                logger.log(Level.INFO, "Dataset updated");
                distributed = true;
                break;

            case UPDATE_IN_PROGRESS:
                Thread.sleep(5000);
                break;

            case UPDATE_FAILED:
                String error = "Dataset distribution failed: " +
datasetDescription.statusAsString() + " "
                    + datasetDescription.statusMessage() + " " + datasetArn;
                logger.log(Level.SEVERE, error);
                break;

            default:
                String unexpectedError = "Unexpected distribution state: " +
datasetDescription.statusAsString() + " "
                    + datasetDescription.statusMessage() + " " + datasetArn;
                logger.log(Level.SEVERE, unexpectedError);

        }

    } while (distributed == false);

    return status;

}

public static void distributeMyDatasetEntries(RekognitionClient rekClient,
String trainingDatasetArn,
String testDatasetArn) throws Exception, RekognitionException {

    try {

        logger.log(Level.INFO, "Distributing {0} dataset to {1} ",
            new Object[] { trainingDatasetArn, testDatasetArn });

        DistributeDataset distributeTrainingDataset =
DistributeDataset.builder().arn(trainingDatasetArn).build();

        DistributeDataset distributeTestDataset =
DistributeDataset.builder().arn(testDatasetArn).build();

        ArrayList<DistributeDataset> datasets = new ArrayList();

        datasets.add(distributeTrainingDataset);
        datasets.add(distributeTestDataset);

        DistributeDatasetEntriesRequest distributeDatasetEntriesRequest =
DistributeDatasetEntriesRequest.builder()
            .datasets(datasets).build();

        rekClient.distributeDatasetEntries(distributeDatasetEntriesRequest);

        DatasetStatus trainingStatus = checkDatasetStatus(rekClient,
trainingDatasetArn);
        DatasetStatus testStatus = checkDatasetStatus(rekClient,
testDatasetArn);

    }

}
```

```
        if (trainingStatus == DatasetStatus.UPDATE_COMPLETE && testStatus ==  
DatasetStatus.UPDATE_COMPLETE) {  
            logger.log(Level.INFO, "Successfully distributed dataset: {0}");  
        } else {  
            throw new Exception("Failed to distribute dataset: " +  
trainingDatasetArn);  
        }  
  
    } catch (RekognitionException e) {  
        logger.log(Level.SEVERE, "Could not distribute dataset: {0}",  
e.getMessage());  
        throw e;  
    }  
}  
  
public static void main(String args[]) {  
  
    String trainingDatasetArn = null;  
    String testDatasetArn = null;  
  
    final String USAGE = "\n" + "Usage: " + "<training_dataset_arn>  
<test_dataset_arn>\n\n" + "Where:\n"  
        + "    training_dataset_arn - the ARN of the dataset that you want  
        to distribute from.\n\n"  
        + "    test_dataset_arn - the ARN of the dataset that you want to  
        distribute to.\n\n";  
  
    if (args.length != 2) {  
        System.out.println(USAGE);  
        System.exit(1);  
    }  
  
    trainingDatasetArn = args[0];  
    testDatasetArn = args[1];  
  
    try {  
  
        // Get the Rekognition client  
        RekognitionClient rekClient = RekognitionClient.builder().build();  
  
        // Distribute the dataset  
        distributeMyDatasetEntries(rekClient, trainingDatasetArn,  
testDatasetArn);  
  
        System.out.println("Datasets distributed.");  
  
        rekClient.close();  
  
    } catch (RekognitionException rekError) {  
        logger.log(Level.SEVERE, "Rekognition client error: {0}",  
rekError.getMessage());  
        System.exit(1);  
    } catch (Exception rekError) {  
        logger.log(Level.SEVERE, "Error: {0}", rekError.getMessage());  
        System.exit(1);  
    }  
}  
}
```

Deleting a dataset

You can delete the training and test datasets from a project.

Topics

- [Deleting a dataset \(Console\) \(p. 238\)](#)
- [Deleting an Amazon Rekognition Custom Labels dataset \(SDK\) \(p. 238\)](#)

Deleting a dataset (Console)

Use the following procedure to delete a dataset. Afterwards, if the project has one remaining dataset (train or test), the project details page is shown. If the project has no remaining datasets, the [Create dataset](#) page is shown.

If you delete the training dataset, you must create a new training dataset for the project before you can train a model. For more information, see [Creating training and test datasets \(Console\) \(p. 59\)](#).

If you delete the test dataset, you can train a model without creating a new test dataset. During training, the training dataset is split to create a new test dataset for the project. Splitting the training dataset reduces the number of images available for training. To maintain quality, we recommend creating a new test dataset before training a model. For more information, see [Adding a dataset to a project \(p. 212\)](#).

To delete a dataset

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. In the left pane, choose **Use Custom Labels**. The Amazon Rekognition Custom Labels landing page is shown.
3. In the left navigation pane, choose **Projects**. The Projects view is shown.
4. Choose the project that contains the dataset that you want to delete.
5. In the left navigation pane, under the project name, choose **Dataset**
6. Choose **Actions**
7. To delete the training dataset, choose **Delete training dataset**.
8. To delete the test dataset, choose **Delete test dataset**.
9. In the **Delete train or test dataset** dialog box, enter **delete** to confirm that you want to delete the dataset.
10. Choose **Delete train or test dataset** to delete the dataset.

Deleting an Amazon Rekognition Custom Labels dataset (SDK)

You delete an Amazon Rekognition Custom Labels dataset by calling [DeleteDataset](#) and supplying the Amazon Resource Name (ARN) of the dataset that you want to delete. To get the ARNs of the training and test datasets within a project, call [DescribeProjects](#). The response includes an array of [ProjectDescription](#) objects. The dataset ARNs (`DatasetArn`) and dataset types (`DatasetType`) are in the `Datasets` list.

If you delete the training dataset, you need to create a new training dataset for the project before you can train a model. If you delete the test dataset, you need to create a new test dataset before you can train the model. For more information, see [Adding a dataset to a project \(SDK\) \(p. 212\)](#).

To delete a dataset (SDK)

1. If you haven't already:

- a. Create or update an IAM user with AmazonRekognitionFullAccess permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
- b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following code to delete a dataset.

AWS CLI

Change the value of `dataset-arn` with the ARN of the dataset that you want to delete.

```
aws rekognition delete-dataset --dataset-arn dataset-arn
```

Python

Use the following code. Supply the following command line parameters:

- `dataset_arn` — the ARN of the dataset that you want to delete.

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import boto3
import argparse
import logging
import time
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def delete_dataset(rek_client, dataset_arn):
    """
    Deletes an Amazon Rekognition Custom Labels dataset.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param dataset_arn: The ARN of the dataset that you want to delete.
    """

    try:
        #Delete the dataset
        logger.info(f"Deleting dataset: {dataset_arn}")

        rek_client.delete_dataset(DatasetArn=dataset_arn)

        deleted=False

        logger.info(f"waiting for dataset deletion {dataset_arn}")

        #dataset might not be deleted yet, so wait.
        while deleted==False:
            try:
                rek_client.describe_dataset(DatasetArn=dataset_arn)
                time.sleep(5)
            except ClientError as err:
                if err.response['Error']['Code'] == 'ResourceNotFoundException':
                    logger.info(f"dataset deleted: {dataset_arn}")
                    deleted=True
                else:
                    raise
    except ClientError as err:
        logger.error(f"Error deleting dataset: {dataset_arn} - {err}")
```

```
logger.info(f"dataset deleted: {dataset_arn}")

return True

except ClientError as err:
    logger.exception(f"Couldn't delete dataset - {dataset_arn}:
{err.response['Error']['Message']}")
    raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "dataset_arn", help="The ARN of the dataset that you want to delete."
    )

def main():
    logging.basicConfig(level=logging.INFO, format"%(levelname)s: %(message)s")
    try:

        #get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Deleting dataset: {args.dataset_arn}")

        #Delete the dataset
        rek_client=boto3.client('rekognition')

        delete_dataset(rek_client,
                      args.dataset_arn)

        print(f"Finished deleting dataset: {args.dataset_arn}")

    except ClientError as err:
        logger.exception(f"Problem deleting dataset: {err}")
        print(f"Problem deleting dataset: {err}")

if __name__ == "__main__":
    main()
```

Java 2

Use the following code. Supply the following command line parameters:

- dataset_arn — the ARN of the dataset that you want to delete.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)
```

```
import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.DeleteDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.DeleteDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

public class DeleteDataset {

    public static final Logger logger =
    Logger.getLogger(DeleteDataset.class.getName());

    public static void deleteMyDataset(RekognitionClient rekClient, String
datasetArn) throws InterruptedException {

        try {

            logger.log(Level.INFO, "Deleting dataset: {0}", datasetArn);

            // Delete the dataset

            DeleteDatasetRequest deleteDatasetRequest =
DeleteDatasetRequest.builder().datasetArn(datasetArn).build();

            DeleteDatasetResponse response =
rekClient.deleteDataset(deleteDatasetRequest);

            // Wait until deletion finishes

            DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder().datasetArn(datasetArn)
                .build();

            Boolean deleted = false;

            do {

                try {

                    rekClient.describeDataset(describeDatasetRequest);
                    Thread.sleep(5000);
                } catch (RekognitionException e) {
                    String errorCode = e.awsErrorDetails().errorCode();
                    if (errorCode.equals("ResourceNotFoundException")) {
                        logger.log(Level.INFO, "Dataset deleted: {}", datasetArn);
                        deleted = true;
                    } else {
                        logger.log(Level.SEVERE, "Client error occurred: {0}",
e.getMessage());
                        throw e;
                    }
                }

            } while (Boolean.FALSE.equals(deleted));

            logger.log(Level.INFO, "Dataset deleted: {0} ", datasetArn);

        } catch (
RekognitionException e) {
            logger.log(Level.SEVERE, "Client error occurred: {0}", e.getMessage());
            throw e;
        }
    }
}
```

```
        }

    }

    public static void main(String args[]) {
        final String USAGE = "\n" + "Usage: " + "<dataset_arn>\n\n" + "Where:\n" +
            "    dataset_arn - The ARN of the dataset that you want to delete.\n\n";
        if (args.length != 1) {
            System.out.println(USAGE);
            System.exit(1);
        }
        String datasetArn = args[0];
        try {
            RekognitionClient rekClient = RekognitionClient.builder().build();

            // Delete the dataset
            deleteMyDataset(rekClient, datasetArn);

            System.out.println(String.format("Dataset deleted: %s", datasetArn));
            rekClient.close();
        } catch (RekognitionException rekError) {
            logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
            System.exit(1);
        }
        catch (InterruptedException intError) {
            logger.log(Level.SEVERE, "Exception while sleeping: {0}",
intError.getMessage());
            System.exit(1);
        }
    }
}
```

Creating a manifest file

You can create a test or training dataset by importing a SageMaker Ground Truth format manifest file. If your images are labeled in a format that isn't a SageMaker Ground Truth manifest file, use the following information to create a SageMaker Ground Truth format manifest file.

Manifest files are in [JSON lines](#) format where each line is a complete JSON object representing the labeling information for an image. Amazon Rekognition Custom Labels supports SageMaker Ground Truth manifests with JSON lines in the following formats:

- [Classification Job Output](#) – Use to add image-level labels to an image. An image-level label defines the class of scene, concept, or object (if object location information isn't needed) that's on an image. An image can have more than one image-level label. For more information, see [Image-Level labels in manifest files \(p. 249\)](#).
- [Bounding Box Job Output](#) – Use to label the class and location of one or more objects on an image. For more information, see [Object localization in manifest files \(p. 252\)](#).

Image-level and localization (bounding-box) JSON lines can be chained together in the same manifest file.

Note

The JSON line examples in this section are formatted for readability.

When you import a manifest file, Amazon Rekognition Custom Labels applies validation rules for limits, syntax, and semantics. For more information, see [Validation rules for manifest files \(p. 255\)](#).

The images referenced by a manifest file must be located in the same Amazon S3 bucket. The manifest file can be located in a different Amazon S3 bucket than the Amazon S3 bucket that stores the images. You specify the location of an image in the `source-ref` field of a JSON line.

Amazon Rekognition needs permissions to access the Amazon S3 bucket where your images are stored. If you are using the console bucket set up for you by Amazon Rekognition Custom Labels, the required permissions are already set up. If you are not using the console bucket, see [Accessing external Amazon S3 Buckets \(p. 7\)](#).

Creating a dataset with a manifest file

The following procedure creates a project with a training and test dataset. The datasets are created from training and test manifest files that you create.

To create a dataset using a SageMaker Ground Truth format manifest file (console)

1. In the console bucket, [create a folder](#) to hold your manifest files.
2. In the console bucket, create a folder to hold your images.
3. Upload your images to the folder you just created.
4. Create a SageMaker Ground Truth format manifest file for your training dataset. For more information, see [Image-Level labels in manifest files \(p. 249\)](#) and [Object localization in manifest files \(p. 252\)](#).

Important

The `source-ref` field value in each JSON line must map to an image that you uploaded.

5. Create an SageMaker Ground Truth format manifest file for your test dataset.
6. [Upload your manifest files](#) to the folder that you just created.
7. Sign in to the AWS Management Console and open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
8. In the left pane, choose **Use Custom Labels**. The Amazon Rekognition Custom Labels landing page is shown.
9. The Amazon Rekognition Custom Labels landing page, choose **Get started**.
10. In the left pane, Choose **Projects**.
11. Choose **Create Project**.
12. In **Project name**, enter a name for your project.
13. Choose **Create project** to create your project.
14. Choose **Create dataset**. The **Create dataset** page is shown.
15. In **Starting configuration**, choose **Start with a training dataset and a test dataset**.
16. In the **Training dataset details** section, choose **Import images labeled by SageMaker Ground Truth**.
17. In **.manifest file location** enter the Amazon S3 location of the training manifest that you uploaded in step 6.
18. In the **Test dataset details** section, choose **Import images labeled by SageMaker Ground Truth**.

19. In **.manifest file location** enter the Amazon S3 location of the test manifest file that uploaded in step 6.
20. Choose **Create Datasets**.
21. Train the model. For more information, see [Training an Amazon Rekognition Custom Labels model \(p. 106\)](#).

Creating a dataset with a manifest file (SDK)

To create a dataset with a manifest file, use the `CreateDataset` API.

To create a dataset with a manifest file(SDK)

- Use the following example code to create the dataset.

AWS CLI

Use the following code to create a dataset. Supply the following command line options:

- `project_arn` — the ARN of the project that you want to add the test dataset to.
- `type` — the type of dataset that you want to create (train or test)
- `bucket` — the bucket that contains the manifest file for the dataset.
- `manifest_file` — the path and file name of the manifest file.

```
aws rekognition create-dataset --project-arn project_arn \  
  --dataset-type type \  
  --dataset-source '{ "GroundTruthManifest": { "S3Object": { "Bucket": "bucket",  
  "Name": "manifest_file" } } }' \  
  
```

Python

The following example creates a dataset.

Use the following values to create the training dataset. Supply the following command line parameters:

- `project_arn` — the ARN of the project that you want to add the test dataset to.
- `dataset_type` — the type of dataset that you want to create (train or test)
- `bucket` — the bucket that contains the manifest file for the dataset.
- `manifest_file` — the path and file name of the manifest file.

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.  
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)  
  
import boto3  
import argparse  
import logging  
import time  
import json  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)  
  
def create_dataset(rek_client, project_arn, dataset_type, bucket, manifest_file):
```

```
"""
Creates an Amazon Rekognition Custom Labels dataset.
:param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
:param project_arn: The ARN of the project in which you want to create a
dataset.
:param dataset_type: The type of the dataset that you want to create (train or
test).
:param bucket: The S3 bucket that contains the manifest file.
:param manifest_file: The path and filename of the manifest file.
"""

try:
    #Create the project
    logger.info(f"Creating {dataset_type} dataset for project {project_arn}")

    dataset_type = dataset_type.upper()

    dataset_source = json.loads(
        '{ "GroundTruthManifest": { "S3Object": { "Bucket": "'
        + bucket
        + '", "Name": "'
        + manifest_file
        + '" } } }'
    )

    response = rek_client.create_dataset(
        ProjectArn=project_arn, DatasetType=dataset_type,
        DatasetSource=dataset_source
    )

    dataset_arn=response['DatasetArn']

    logger.info(f"dataset ARN: {dataset_arn}")

    finished=False
    while finished==False:

        dataset=rek_client.describe_dataset(DatasetArn=dataset_arn)

        status=dataset['DatasetDescription']['Status']

        if status == "CREATE_IN_PROGRESS":

            logger.info((f"Creating dataset: {dataset_arn} "))
            time.sleep(5)
            continue

        if status == "CREATE_COMPLETE":
            logger.info(f"Dataset created: {dataset_arn}")
            finished=True
            continue

        if status == "CREATE_FAILED":
            logger.exception(f"Dataset creation failed: {status} : {dataset_arn}")
            raise Exception (f"Dataset creation failed: {status} : {dataset_arn}")

            logger.exception(f"Failed. Unexpected state for dataset creation: {status} : {dataset_arn}")
            raise Exception(f"Failed. Unexpected state for dataset creation: {status} : {dataset_arn}")

    return dataset_arn
```

```
        except ClientError as err:
            logger.exception(f"Couldn't create dataset: {err.response['Error']}")
            raise

    def add_arguments(parser):
        """
        Adds command line arguments to the parser.
        :param parser: The command line parser.
        """

        parser.add_argument(
            "project_arn", help="The ARN of the project in which you want to create the
dataset.")
        )

        parser.add_argument(
            "dataset_type", help="The type of the dataset that you want to create
(train or test).")
        )

        parser.add_argument(
            "bucket", help="The S3 bucket that contains the manifest file.")
        )

        parser.add_argument(
            "manifest_file", help="The path and filename of the manifest file.")
        )

    def main():

        logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

        try:

            #get command line arguments
            parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
            add_arguments(parser)
            args = parser.parse_args()

            print(f"Creating {args.dataset_type} dataset for project
{args.project_arn}")

            #Create the project
            rek_client=boto3.client('rekognition')

            dataset_arn=create_dataset(rek_client,
            args.project_arn,
            args.dataset_type,
            args.bucket,
            args.manifest_file)

            print(f"Finished creating dataset: {dataset_arn}")

        except ClientError as err:
            logger.exception(f"Problem creating dataset: {err}")
            print(f"Problem creating dataset: {err}")
        except Exception as err:
            logger.exception(f"Problem creating dataset: {err}")
            print(f"Problem creating dataset: {err}")

    if __name__ == "__main__":
```

```
main()
```

Java 2

The following example creates a dataset.

Use the following values to create the training dataset. Supply the following command line parameters:

- `project_arn` — the ARN of the project that you want to add the test dataset to.
- `dataset_type` — the type of dataset that you want to create (train or test)
- `bucket` — the bucket that contains the manifest file for the dataset.
- `manifest_file` — the path and file name of the manifest file.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.CreateDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.CreateDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.CreateProjectRequest;
import software.amazon.awssdk.services.rekognition.model.CreateProjectResponse;
import software.amazon.awssdk.services.rekognition.model.DatasetDescription;
import software.amazon.awssdk.services.rekognition.model.DatasetSource;
import software.amazon.awssdk.services.rekognition.model.DatasetStatus;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetRequest;
import software.amazon.awssdk.services.rekognition.model.DescribeDatasetResponse;
import software.amazon.awssdk.services.rekognition.model.DescribeProjectsRequest;
import software.amazon.awssdk.services.rekognition.model.DescribeProjectsResponse;
import software.amazon.awssdk.services.rekognition.model.ProjectDescription;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.util.List;
import java.util.Objects;
import java.util.logging.Level;
import java.util.logging.Logger;

public class CreateDataset {

    public static final Logger logger =
    Logger.getLogger(CreateDataset.class.getName());

    public static String createMyDataset(RekognitionClient rekClient, String
    projectArn, String datasetArn) {

        try {
            logger.log(Level.INFO, "Creating dataset for project : {0} from dataset
            {1} ",
            new Object[] {projectArn,datasetArn});

            DatasetSource datasetSource = DatasetSource.builder()
                .datasetArn(datasetArn).build();

            CreateDatasetRequest createDatasetRequest =
            CreateDatasetRequest.builder()
                .datasetSource(datasetSource).build();
        }
    }
}
```

```
        CreateDatasetResponse response =
rekClient.createDataset(createDatasetRequest);

        Boolean deleted = false;

        do {

            DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder()
                .datasetArn(response.datasetArn())
                .build();
            DescribeDatasetResponse describeDatasetResponse =
rekClient.describeDataset(describeDatasetRequest);

            DatasetStatus status =
describeDatasetResponse.datasetDescription().status();

            switch (status) {

                case (DatasetStatus) DatasetStatus.CREATE_COMPLETE:
                    logger.log(Level.INFO, "Dataset created");

                }

            deleted = true;

            for (ProjectDescription projectDescription : projectDescriptions) {

                if (Objects.equals(projectDescription.projectArn(),
projectArn)) {
                    deleted = false;
                    logger.log(Level.INFO, "Not deleted: {}",

projectDescription.projectArn());
                    Thread.sleep(5000);
                    break;

                }

            }

        } while (Boolean.FALSE.equals(deleted));

        logger.log(Level.INFO, "Project ARN: {} ", response.projectArn());

        return response.projectArn();

    } catch (RekognitionException e) {
        logger.log(Level.SEVERE, "Could not create project: {}",
e.getMessage());
        throw e;
    }

}

public static void main(String args[]) {

    final String USAGE = "\n" + "Usage: " + "<project_name> <bucket> <image>\n"
"\n" + "Where:\n" +
        "    project_name - A name for the new project\n\n";

    if (args.length != 1) {
        System.out.println(USAGE);
        System.exit(1);
    }

    String projectName = args[0];
```

```
String projectArn = null;
;

try {
    // Get the Rekognition client
    RekognitionClient rekClient = RekognitionClient.builder().build();

    // Create the project
    projectArn = createMyProject(rekClient, projectName);

    System.out.println(String.format("Created project: %s %nProject ARN: %s",
        projectName, projectArn));

    rekClient.close();
} catch (RekognitionException rekError) {
    logger.log(Level.SEVERE, "Rekognition client error: {0}",
        rekError.getMessage());
    System.exit(1);
}
}
```

Image-Level labels in manifest files

To import image-level labels (images labeled with scenes, concepts, or objects that don't require localization information), you add SageMaker Ground Truth [Classification Job Output](#) format JSON lines to a manifest file. A manifest file is made of one or more JSON lines, one for each image that you want to import.

Tip

To simplify creation of a manifest file, we provide a Python script that creates a manifest file from a CSV file. For more information, see [Creating a manifest file from a CSV file \(p. 313\)](#).

To create a manifest file for image-level labels

1. Create an empty text file.
2. Add a JSON line for each image that you want to import. Each JSON line should look similar to the following.

```
{"source-ref":"s3://custom-labels-console-us-east-1-nnnnnnnnnn/gt-job/manifest/IMG_1133.png","TestCLConsoleBucket":0,"TestCLConsoleBucket-metadata":{"confidence":0.95,"job-name":"labeling-job/testclconsolebucket","class-name":"Echo Dot","human-annotated":"yes","creation-date":"2020-04-15T20:17:23.433061","type":"groundtruth/image-classification"}}
```

3. Save the file. You can use the extension `.manifest`, but it is not required.
4. Create a dataset using the manifest file that you created. For more information, see [To create a dataset using a SageMaker Ground Truth format manifest file \(console\) \(p. 243\)](#).

Image-Level JSON Lines

In this section, we show you how to create a JSON line for a single image. Consider the following image. A scene for the following image might be called *Sunrise*.



The JSON line for the preceding image, with the scene *Sunrise*, might be the following.

```
{  
  "source-ref": "s3://bucket/images/sunrise.png",  
  "testdataset-classification_Sunrise": 1,  
  "testdataset-classification_Sunrise-metadata": {  
    "confidence": 1,  
    "job-name": "labeling-job/testdataset-classification_Sunrise",  
    "class-name": "Sunrise",  
    "human-annotated": "yes",  
    "creation-date": "2020-03-06T17:46:39.176",  
    "type": "groundtruth/image-classification"  
}
```

}

Note the following information.

source-ref

(Required) The Amazon S3 location of the image. The format is "s3://*BUCKET/OBJECT_PATH*". Images in an imported dataset must be stored in the same Amazon S3 bucket.

testdataset-classification_Sunrise

(Required) The label attribute. You choose the field name. The field value (1 in the preceding example) is a label attribute identifier. It is not used by Amazon Rekognition Custom Labels and can be any integer value. There must be corresponding metadata identified by the field name with *-metadata* appended. For example, "testdataset-classification_Sunrise-metadata".

testdataset-classification_Sunrise-metadata

(Required) Metadata about the label attribute. The field name must be the same as the label attribute with *-metadata* appended.

confidence

(Required) Currently not used by Amazon Rekognition Custom Labels but a value between 0 and 1 must be supplied.

job-name

(Optional) A name that you choose for the job that processes the image.

class-name

(Required) A class name that you choose for the scene or concept that applies to the image. For example, "Sunrise".

human-annotated

(Required) Specify "yes", if the annotation was completed by a human. Otherwise "no".

creation-date

(Required) The Coordinated Universal Time (UTC) date and time that the label was created.

type

(Required) The type of processing that should be applied to the image. For image-level labels, the value is "groundtruth/image-classification".

Adding multiple image-level labels to an image

You can add multiple labels to an image. For example, the follow JSON adds two labels, *football* and *ball* to a single image.

```
{  
  "source-ref": "S3 bucket location",  
  "sport0": 0, # FIRST label  
  "sport0-metadata": {  
    "class-name": "football",  
    "confidence": 0.8,  
    "type": "groundtruth/image-classification",  
  },  
  "sport1": 1, # SECOND label  
  "sport1-metadata": {  
    "class-name": "ball",  
    "confidence": 0.9,  
    "type": "groundtruth/image-classification",  
  },  
}
```

```
        "job-name": "identify-sport",
        "human-annotated": "yes",
        "creation-date": "2018-10-18T22:18:13.527256"
    },
    "sport1":1, # SECOND label
    "sport1-metadata": {
        "class-name": "ball",
        "confidence": 0.8,
        "type": "groundtruth/image-classification",
        "job-name": "identify-sport",
        "human-annotated": "yes",
        "creation-date": "2018-10-18T22:18:13.527256"
    }
} # end of annotations for 1 image
```

Object localization in manifest files

You can import images labeled with object localization information by adding SageMaker Ground Truth [Bounding Box Job Output](#) format JSON lines to a manifest file.

Localization information represents the location of an object on an image. The location is represented by a bounding box that surrounds the object. The bounding box structure contains the upper-left coordinates of the bounding box and the bounding box's width and height. A bounding box format JSON line includes bounding boxes for the locations of one or more objects on an image and the class of each object on the image.

A manifest file is made of one or more JSON lines, each line contains the information for a single image.

To create a manifest file for object localization

1. Create an empty text file.
2. Add a JSON line for each image the that you want to import. Each JSON line should look similar to the following.

```
{"source-ref": "s3://bucket/images/IMG_1186.png", "bounding-box": {"image_size": [{"width": 640, "height": 480, "depth": 3}], "annotations": [{"class_id": 1, "top": 251, "left": 399, "width": 155, "height": 101}, {"class_id": 0, "top": 65, "left": 86, "width": 220, "height": 334}]], "bounding-box-metadata": {"objects": [{"confidence": 1}, {"confidence": 1}], "class-map": {"0": "Echo", "1": "Echo Dot"}, "type": "groundtruth/object-detection", "human-annotated": "yes", "creation-date": "2013-11-18T02:53:27", "job-name": "my job"}}
```

3. Save the file. You can use the extension `.manifest`, but it is not required.
4. Create a dataset using the file that you just created. For more information, see [To create a dataset using a SageMaker Ground Truth format manifest file \(console\) \(p. 243\)](#).

Object bounding Box JSON lines

In this section, we show you how to create a JSON line for a single image. The following image shows bounding boxes around Amazon Echo and Amazon Echo Dot devices.



The following is the bounding box JSON line for the preceding image.

```
{  
  "source-ref": "s3://custom-labels-bucket/images/IMG_1186.png",  
  "bounding-box": {  
    "image_size": [{  
      "width": 640,  
      "height": 480,  
      "depth": 3  
    }],  
    "annotations": [{  
      "class_id": 1,  
      "top": 251,  
      "left": 399,  
      "width": 155,  
      "height": 101  
    }, {  
      "class_id": 0,  
      "top": 65,  
      "left": 86,  
      "width": 220,  
      "height": 334  
    }]  
  },  
  "bounding-box-metadata": {  
    "objects": [{  
      "confidence": 1  
    }, {  
      "confidence": 1  
    }],  
    "class-map": {  
      "0": "Echo",  
      "1": "Echo Dot"  
    },  
    "type": "groundtruth/object-detection",  
    "human-annotated": "yes",  
    "creation-date": "2013-11-18T02:53:27",  
    "job-name": "my job"  
  }  
}
```

Note the following information.

source-ref

(Required) The Amazon S3 location of the image. The format is "s3://*BUCKET/OBJECT_PATH*". Images in an imported dataset must be stored in the same Amazon S3 bucket.

bounding-box

(Required) The label attribute. You choose the field name. Contains the image size and the bounding boxes for each object detected in the image. There must be corresponding metadata identified by the field name with *-metadata* appended. For example, "bounding-box-metadata".

image_size

(Required) A single element array containing the size of the image in pixels.

- *height* – (Required) The height of the image in pixels.
- *width* – (Required) The depth of the image in pixels.
- *depth* – (Required) The number of channels in the image. For RGB images, the value is 3. Not currently used by Amazon Rekognition Custom Labels, but a value is required.

annotations

(Required) An array of bounding box information for each object detected in the image.

- *class_id* – (Required) Maps to the label in *class-map*. In the preceding example, the object with the *class_id* of 1 is the Echo Dot in the image.
- *top* – (Required) The distance from the top of the image to the top of the bounding box, in pixels.
- *left* – (Required) The distance from the left of the image to the left of the bounding box, in pixels.
- *width* – (Required) The width of the bounding box, in pixels.
- *height* – (Required) The height of the bounding box, in pixels.

bounding-box-metadata

(Required) Metadata about the label attribute. The field name must be the same as the label attribute with *-metadata* appended. An array of bounding box information for each object detected in the image.

Objects

(Required) An array of objects that are in the image. Maps to the *annotations* array by index. The confidence attribute isn't used by Amazon Rekognition Custom Labels.

class-map

(Required) A map of the classes that apply to objects detected in the image.

type

(Required) The type of classification job. "groundtruth/object-detection" identifies the job as object detection.

creation-date

(Required) The Coordinated Universal Time (UTC) date and time that the label was created.

human-annotated

(Required) Specify "yes", if the annotation was completed by a human. Otherwise "no".

job-name

(Optional) The name of the job that processes the image.

Validation rules for manifest files

When you import a manifest file, Amazon Rekognition Custom Labels applies validation rules for limits, syntax, and semantics. The SageMaker Ground Truth schema enforces syntax validation. For more information, see [Output](#). The following are the validation rules for limits and semantics.

Note

- The 20% invalidity rules apply cumulatively across all validation rules. If the import exceeds the 20% limit due to any combination, such as 15% invalid JSON and 15% invalid images, the import fails.
- Each dataset object is a line in the manifest. Blank/invalid lines are also counted as dataset objects.
- Overlaps are (common labels between test and train)/(train labels).

Topics

- [Limits \(p. 255\)](#)
- [Semantics \(p. 255\)](#)

Limits

Validation	Limit	Error raised
Manifest file size	Maximum 1 GB	Error
Maximum line count for a manifest file	Maximum of 250,000 dataset objects as lines in a manifest.	Error
Lower boundary on total number of valid dataset objects per label	≥ 1	Error
Lower boundary on labels	≥ 2	Error
Upper bound on labels	≤ 250	Error
Minimum bounding boxes per image	0	None
Maximum bounding boxes per image	50	None

Semantics

Validation	Limit	Error raised
Empty manifest		Error
Missing/in-accessible source-ref object	Number of objects less than 20%	Warning
Missing/in-accessible source-ref object	Number of objects $> 20\%$	Error

Validation	Limit	Error raised
Test labels not present in training dataset	At least 50% overlap in the labels	Error
Mix of label vs. object examples for same label in a dataset. Classification and detection for the same class in a dataset object.		No error or warning
Overlapping assets between test and train	There should not be an overlap between test and training datasets.	
Images in a dataset must be from same bucket	Error if the objects are in a different bucket	Error

Transforming COCO datasets

[COCO](#) is a format for specifying large-scale object detection, segmentation, and captioning datasets. This Python [example \(p. 262\)](#) shows you how to transform a COCO object detection format dataset into an Amazon Rekognition Custom Labels [bounding box format manifest file \(p. 252\)](#). This section also includes information that you can use to write your own code.

A COCO format JSON file consists of five sections providing information for *an entire dataset*. For more information, see [COCO format \(p. 260\)](#).

- [info](#) – general information about the dataset.
- [licenses](#) – license information for the images in the dataset.
- [images \(p. 261\)](#) – a list of images in the dataset.
- [annotations \(p. 261\)](#) – a list of annotations (including bounding boxes) that are present in all images in the dataset.
- [categories \(p. 262\)](#) – a list of label categories.

You need information from the [images](#), [annotations](#), and [categories](#) lists to create an Amazon Rekognition Custom Labels manifest file.

An Amazon Rekognition Custom Labels manifest file is in JSON lines format where each line has the bounding box and label information for one or more objects *on an image*. For more information, see [Object localization in manifest files \(p. 252\)](#).

Mapping COCO Objects to a Custom Labels JSON Line

To transform a COCO format dataset, you map the COCO dataset to an Amazon Rekognition Custom Labels manifest file for object localization. For more information, see [Object localization in manifest files \(p. 252\)](#). To build a JSON line for each image, the manifest file needs to map the COCO dataset image, annotation, and category object field IDs.

The following is an example COCO manifest file. For more information, see [COCO format \(p. 260\)](#).

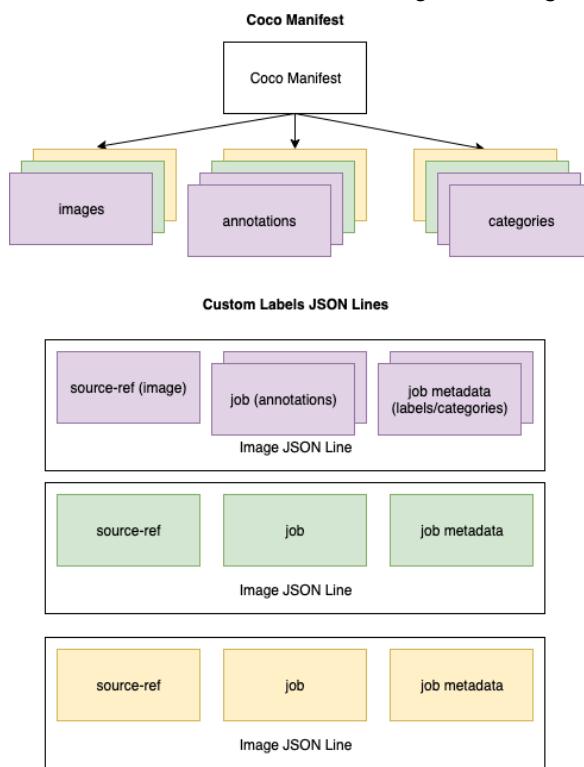
```
{
  "info": {
    "description": "COCO 2017 Dataset", "url": "http://cocodataset.org", "version": "1.0", "year": 2017, "contributor": "COCO Consortium", "date_created": "2017/09/01"
  }
}
```

```

    ],
    "licenses": [
        {"url": "http://creativecommons.org/licenses/by/2.0/","id": 4,"name": "Attribution License"}
    ],
    "images": [
        {"id": 242287, "license": 4, "coco_url": "http://images.cocodataset.org/val2017/xxxxxxxxxxxxxx.jpg", "flickr_url": "http://farm3.staticflickr.com/2626/xxxxxxxxxxxx.jpg", "width": 426, "height": 640, "file_name": "xxxxxxxxxx.jpg", "date_captured": "2013-11-15 02:41:42"},
        {"id": 245915, "license": 4, "coco_url": "http://images.cocodataset.org/val2017/nnnnnnnnnnnnn.jpg", "flickr_url": "http://farm1.staticflickr.com/88/xxxxxxxxxxxx.jpg", "width": 640, "height": 480, "file_name": "nnnnnnnnnnn.jpg", "date_captured": "2013-11-18 02:53:27"}
    ],
    "annotations": [
        {"id": 125686, "category_id": 0, "iscrowd": 0, "segmentation": [[164.81, 417.51, ..., 167.55, 410.64]], "image_id": 242287, "area": 42061.80340000001, "bbox": [19.23, 383.18, 314.5, 244.46]},
        {"id": 1409619, "category_id": 0, "iscrowd": 0, "segmentation": [[376.81, 238.8, ..., 382.74, 241.17]], "image_id": 245915, "area": 3556.2197000000015, "bbox": [399, 251, 155, 101]},
        {"id": 1410165, "category_id": 1, "iscrowd": 0, "segmentation": [[486.34, 239.01, ..., 495.95, 244.39]], "image_id": 245915, "area": 1775.893249999994, "bbox": [86, 65, 220, 334]}
    ],
    "categories": [
        {"supercategory": "speaker","id": 0,"name": "echo"},
        {"supercategory": "speaker","id": 1,"name": "echo dot"}
    ]
}
}

```

The following diagram shows how the COCO dataset lists for a *dataset* map to Amazon Rekognition Custom Labels JSON lines for an *image*. Matching colors indicate information for a single image.



To get the COCO objects for a single JSON line

1. For each image in the images list, get the annotation from the annotations list where the value of the annotation field `image_id` matches the image `id` field.
2. For each annotation matched in step 1, read through the `categories` list and get each `category` where the value of the `category` field `id` matches the annotation object `category_id` field.
3. Create a JSON line for the image using the matched `image`, `annotation`, and `category` objects. To map the fields, see [Mapping COCO object fields to a Custom Labels JSON line object fields \(p. 258\)](#).
4. Repeat steps 1–3 until you have created JSON lines for each `image` object in the `images` list.

For example code, see [Transforming a COCO dataset \(p. 262\)](#).

Mapping COCO object fields to a Custom Labels JSON line object fields

After you identify the COCO objects for an Amazon Rekognition Custom Labels JSON line, you need to map the COCO object fields to the respective Amazon Rekognition Custom Labels JSON line object fields. The following example Amazon Rekognition Custom Labels JSON line maps one image (`id=000000245915`) to the preceding COCO JSON example. Note the following information.

- `source-ref` is the location of the image in an Amazon S3 bucket. If your COCO images aren't stored in an Amazon S3 bucket, you need to move them to an Amazon S3 bucket.
- The `annotations` list contains an annotation object for each object on the image. An annotation object includes bounding box information (`top`, `left`, `width`, `height`) and a label identifier (`class_id`).
- The label identifier (`class_id`) maps to the `class-map` list in the `metadata`. It lists the labels used on the image.

```
{  
  "source-ref": "s3://custom-labels-bucket/images/000000245915.jpg",  
  "bounding-box": {  
    "image_size": {  
      "width": 640,  
      "height": 480,  
      "depth": 3  
    },  
    "annotations": [{  
      "class_id": 0,  
      "top": 251,  
      "left": 399,  
      "width": 155,  
      "height": 101  
    }, {  
      "class_id": 1,  
      "top": 65,  
      "left": 86,  
      "width": 220,  
      "height": 334  
    }]  
  },  
  "bounding-box-metadata": {  
    "objects": [{  
      "confidence": 1  
    }, {  
      "confidence": 1  
    }],  
    "class-map": {  
      "0": "Echo",  
      "1": "Chair"  
    }  
  }  
}
```

```
    "1": "Echo Dot"
},
"type": "groundtruth/object-detection",
"human-annotated": "yes",
"creation-date": "2018-10-18T22:18:13.527256",
"job-name": "my job"
}
}
```

Use the following information to map Amazon Rekognition Custom Labels manifest file fields to COCO dataset JSON fields.

source-ref

The S3 format URL for the location of the image. The image must be stored in an S3 bucket. For more information, see [source-ref \(p. 254\)](#). If the `coco_url` COCO field points to an S3 bucket location, you can use the value of `coco_url` for the value of `source-ref`. Alternatively, you can map `source-ref` to the `file_name` (COCO) field and in your transform code, add the required S3 path to where the image is stored.

bounding-box

A label attribute name of your choosing. For more information, see [bounding-box \(p. 254\)](#).

image_size

The size of the image in pixels. Maps to an `image` object in the [images \(p. 261\)](#) list.

- `height-> image (p. 261).height`
- `width-> image (p. 261).width`
- `depth-> Not used by Amazon Rekognition Custom Labels but a value must be supplied.`

annotations

A list of annotation objects. There's one annotation for each object on the image.

annotation

Contains bounding box information for one instance of an object on the image.

- `class_id -> numerical id mapping to Custom Label's class-map list.`
- `top -> bbox (p. 261)[1]`
- `left -> bbox (p. 261)[0]`
- `width -> bbox (p. 261)[2]`
- `height -> bbox (p. 261)[3]`

bounding-box-metadata

Metadata for the `label` attribute. Includes the labels and label identifiers. For more information, see [bounding-box-metadata \(p. 254\)](#).

Objects

An array of objects in the image. Maps to the `annotations` list by index.

Object

- **confidence**->Not used by Amazon Rekognition Custom Labels, but a value (1) is required.

class-map

A map of the labels (classes) that apply to objects detected in the image. Maps to category objects in the [categories \(p. 262\)](#) list.

- **id** -> [category \(p. 262\)](#).id
- **id value** -> [category \(p. 262\)](#).name

type

Must be groundtruth/object-detection

human-annotated

Specify yes or no. For more information, see [bounding-box-metadata \(p. 254\)](#).

creation-date -> [image \(p. 261\)](#).date_captured

The creation date and time of the image. Maps to the [image \(p. 261\)](#).date_captured field of an image in the COCO images list. Amazon Rekognition Custom Labels expects the format of creation-date to be Y-M-DTH:M:S.

job-name

A job name of your choosing.

COCO format

A COCO dataset consists of five sections of information that provide information for the entire dataset. The format for a COCO object detection dataset is documented at [COCO Data Format](#).

- **info** – general information about the dataset.
- **licenses** – license information for the images in the dataset.
- [images \(p. 261\)](#) – a list of images in the dataset.
- [annotations \(p. 261\)](#) – a list of annotations (including bounding boxes) that are present in all images in the dataset.
- [categories \(p. 262\)](#) – a list of label categories.

To create a Custom Labels manifest, you use the images, annotations, and categories lists from the COCO manifest file. The other sections (info, licences) aren't required. The following is an example COCO manifest file.

```
{  
  "info": {  
    "description": "COCO 2017 Dataset", "url": "http://cocodataset.org", "version":  
    "1.0", "year": 2017, "contributor": "COCO Consortium", "date_created": "2017/09/01"  
  },  
  "licenses": [  
    {"url": "http://creativecommons.org/licenses/by/2.0/", "id": 4, "name": "Attribution  
    License"}  
  ],  
  "images": [  
    {"id": 242287, "license": 4, "coco_url": "http://images.cocodataset.org/val2017/  
    xxxxxxxxxxxx.jpg", "flickr_url": "http://farm3.staticflickr.com/2626/xxxxxxxxxx.jpg",  
  ]  
}
```

```
    "width": 426, "height": 640, "file_name": "xxxxxxxxxx.jpg", "date_captured": "2013-11-15 02:41:42"},  
    {"id": 245915, "license": 4, "coco_url": "http://images.cocodataset.org/val2017/nnnnnnnnnnnnnnnnn.jpg", "flickr_url": "http://farm1.staticflickr.com/88/xxxxxxxxxxxxxx.jpg", "width": 640, "height": 480, "file_name": "nnnnnnnnnnn.jpg", "date_captured": "2013-11-18 02:53:27"}  
],  
    "annotations": [  
        {"id": 125686, "category_id": 0, "iscrowd": 0, "segmentation": [[164.81, 417.51, ..., 167.55, 410.64]], "image_id": 242287, "area": 42061.80340000001, "bbox": [19.23, 383.18, 314.5, 244.46]},  
        {"id": 1409619, "category_id": 0, "iscrowd": 0, "segmentation": [[376.81, 238.8, ..., 382.74, 241.17]], "image_id": 245915, "area": 3556.2197000000015, "bbox": [399, 251, 155, 101]},  
        {"id": 1410165, "category_id": 1, "iscrowd": 0, "segmentation": [[486.34, 239.01, ..., 495.95, 244.39]], "image_id": 245915, "area": 1775.893249999994, "bbox": [86, 65, 220, 334]}  
],  
    "categories": [  
        {"supercategory": "speaker", "id": 0, "name": "echo"},  
        {"supercategory": "speaker", "id": 1, "name": "echo dot"}  
]  
}
```

images list

The images referenced by a COCO dataset are listed in the images array. Each image object contains information about the image such as the image file name. In the following example image object, note the following information and which fields are required to create an Amazon Rekognition Custom Labels manifest file.

- **id** – (Required) A unique identifier for the image. The **id** field maps to the **id** field in the annotations array (where bounding box information is stored).
- **license** – (Not Required) Maps to the license array.
- **coco_url** – (Optional) The location of the image.
- **flickr_url** – (Not required) The location of the image on Flickr.
- **width** – (Required) The width of the image.
- **height** – (Required) The height of the image.
- **file_name** – (Required) The image file name. In this example, **file_name** and **id** match, but this is not a requirement for COCO datasets.
- **date_captured** –(Required) the date and time the image was captured.

```
{  
    "id": 245915,  
    "license": 4,  
    "coco_url": "http://images.cocodataset.org/val2017/nnnnnnnnnnnnnnnnn.jpg",  
    "flickr_url": "http://farm1.staticflickr.com/88/nnnnnnnnnnnnnnnnnnn.jpg",  
    "width": 640,  
    "height": 480,  
    "file_name": "000000245915.jpg",  
    "date_captured": "2013-11-18 02:53:27"  
}
```

annotations (bounding boxes) list

Bounding box information for all objects on all images is stored the annotations list. A single annotation object contains bounding box information for a single object and the object's label on an image. There is an annotation object for each instance of an object on an image.

In the following example, note the following information and which fields are required to create an Amazon Rekognition Custom Labels manifest file.

- **id** – (Not required) The identifier for the annotation.
- **image_id** – (Required) Corresponds to the image **id** in the **images** array.
- **category_id** – (Required) The identifier for the label that identifies the object within a bounding box. It maps to the **id** field of the **categories** array.
- **iscrowd** – (Not required) Specifies if the image contains a crowd of objects.
- **segmentation** – (Not required) Segmentation information for objects on an image. Amazon Rekognition Custom Labels doesn't support segmentation.
- **area** – (Not required) The area of the annotation.
- **bbox** – (Required) Contains the coordinates, in pixels, of a bounding box around an object on the image.

```
{  
  "id": 1409619,  
  "category_id": 1,  
  "iscrowd": 0,  
  "segmentation": [  
    [86.0, 238.8,.....382.74, 241.17]  
  ],  
  "image_id": 245915,  
  "area": 3556.2197000000015,  
  "bbox": [86, 65, 220, 334]  
}
```

categories list

Label information is stored the **categories** array. In the following example **category** object, note the following information and which fields are required to create an Amazon Rekognition Custom Labels manifest file.

- **supercategory** – (Not required) The parent category for a label.
- **id** – (Required) The label identifier. The **id** field maps to the **category_id** field in an annotation object. In the following example, The identifier for an echo dot is 2.
- **name** – (Required) The label name.

```
{"supercategory": "speaker", "id": 2, "name": "echo dot"}
```

Transforming a COCO dataset

Use the following Python example to transform bounding box information from a COCO format dataset into an Amazon Rekognition Custom Labels manifest file. The code uploads the created manifest file to your Amazon S3 bucket. The code also provides an AWS CLI command that you can use to upload your images.

To transform a COCO dataset (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with **AmazonS3FullAccess** permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).

- b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following Python code to transform a COCO dataset. Set the following values.
- `s3_bucket` – The name of the S3 bucket in which you want to store the images and Amazon Rekognition Custom Labels manifest file.
 - `s3_key_path_images` – The path to where you want to place the images within the S3 bucket (`s3_bucket`).
 - `s3_key_path_manifest_file` – The path to where you want to place the Custom Labels manifest file within the S3 bucket (`s3_bucket`).
 - `local_path` – The local path to where the example opens the input COCO dataset and also saves the new Custom Labels manifest file.
 - `local_images_path` – The local path to the images that you want to use for training.
 - `coco_manifest` – The input COCO dataset filename.
 - `cl_manifest_file` – A name for the manifest file created by the example. The file is saved at the location specified by `local_path`. By convention, the file has the extension `.manifest`, but this is not required.
 - `job_name` – A name for the Custom Labels job.

```
import json
import os
import random
import shutil
import datetime
import botocore
import boto3
import PIL.Image as Image
import io

#S3 location for images
s3_bucket = 'bucket'
s3_key_path_manifest_file = 'path to custom labels manifest file/'
s3_key_path_images = 'path to images/'
s3_path='s3://' + s3_bucket + '/' + s3_key_path_images
s3 = boto3.resource('s3')

#Local file information
local_path='path to input COCO dataset and output Custom Labels manifest/'
local_images_path='path to COCO images/'
coco_manifest = 'COCO dataset JSON file name'
coco_json_file = local_path + coco_manifest
job_name='Custom Labels job name'
cl_manifest_file = 'custom_labels.manifest'

label_attribute ='bounding-box'

open(local_path + cl_manifest_file, 'w').close()

# class representing a Custom Label JSON line for an image
class cl_json_line:
    def __init__(self,job, img):

        #Get image info. Annotations are dealt with separately
        sizes=[]
        image_size={}
        image_size["width"] = img["width"]
        image_size["depth"] = 3
        image_size["height"] = img["height"]
        sizes.append(image_size)
```

```
        bounding_box={}
        bounding_box["annotations"] = []
        bounding_box["image_size"] = sizes

        self.__dict__["source-ref"] = s3_path + img['file_name']
        self.__dict__[job] = bounding_box

        #get metadata
        metadata = {}
        metadata['job-name'] = job_name
        metadata['class-map'] = {}
        metadata['human-annotated']='yes'
        metadata['objects'] = []
        date_time_obj = datetime.datetime.strptime(img['date_captured'], '%Y-%m-%d %H:
        %M:%S')
        metadata['creation-date']= date_time_obj.strftime('%Y-%m-%dT%H:%M:%S')
        metadata['type']='groundtruth/object-detection'

        self.__dict__[job + '-metadata'] = metadata

print("Getting image, annotations, and categories from COCO file...")

with open(coco_json_file) as f:

    #Get custom label compatible info
    js = json.load(f)
    images = js['images']
    categories = js['categories']
    annotations = js['annotations']

    print('Images: ' + str(len(images)))
    print('annotations: ' + str(len(annotations)))
    print('categories: ' + str(len(categories)))

print("Creating CL JSON lines...")

images_dict = {image['id']: cl_json_line(label_attribute, image) for image in images}

print('Parsing annotations...')
for annotation in annotations:

    image=images_dict[annotation['image_id']]

    cl_annotation = {}
    cl_class_map={}

    # get bounding box information
    cl_bounding_box={}
    cl_bounding_box['left'] = annotation['bbox'][0]
    cl_bounding_box['top'] = annotation['bbox'][1]

    cl_bounding_box['width'] = annotation['bbox'][2]
    cl_bounding_box['height'] = annotation['bbox'][3]
    cl_bounding_box['class_id'] = annotation['category_id']

    setattr(image, label_attribute)[['annotations']].append(cl_bounding_box)

    for category in categories:
        if annotation['category_id'] == category['id']:
            setattr(image, label_attribute + '-metadata')[['class-map']]
[category['id']] = category['name']
```

```
cl_object={}
cl_object['confidence'] = int(1) #not currently used by Custom Labels
getattr(image, label_attribute + '-metadata')['objects'].append(cl_object)

print('Done parsing annotations')

# Create manifest file.
print('Writing Custom Labels manifest...')

for im in images_dict.values():

    with open(local_path+cl_manifest_file, 'a+') as outfile:
        json.dump(im.__dict__,outfile)
        outfile.write('\n')
        outfile.close()

# Upload manifest file to S3 bucket.
print ('Uploading Custom Labels manifest file to S3 bucket')
print('Uploading' + local_path + cl_manifest_file + ' to ' +
s3_key_path_manifest_file)
print(s3_bucket)
s3 = boto3.resource('s3')
s3.Bucket(s3_bucket).upload_file(local_path + cl_manifest_file,
s3_key_path_manifest_file + cl_manifest_file)

# Print S3 URL to manifest file,
print ('S3 URL Path to manifest file. ')
print('\033[1m s3://'+ s3_bucket + '/' + s3_key_path_manifest_file + cl_manifest_file
+ '\033[0m')

# Display aws s3 sync command.
print ('\nAWS CLI s3 sync command to upload your images to S3 bucket. ')
print ('\033[1m aws s3 sync ' + local_images_path + ' ' + s3_path + '\033[0m')
```

3. Run the code.
4. In the program output, note the `s3 sync` command. You need it in the next step.
5. At the command prompt, run the `s3 sync` command. Your images are uploaded to the S3 bucket. If the command fails during upload, run it again until your local images are synchronized with the S3 bucket.
6. In the program output, note the S3 URL path to the manifest file. You need it in the next step.
7. Follow the instruction at [Creating a manifest file \(p. 242\)](#) to create a dataset with the uploaded manifest file. You don't need to do steps 1-6. For step 14, use the Amazon S3 URL you noted in the previous step.
8. Build your model. For more information, see [Training an Amazon Rekognition Custom Labels model \(p. 106\)](#).

Transforming multi-label SageMaker Ground Truth manifest files

This topic shows you how to transform a multi-label Amazon SageMaker Ground Truth manifest file to an Amazon Rekognition Custom Labels format manifest file.

SageMaker Ground Truth manifest files for multi-label jobs are formatted differently than Amazon Rekognition Custom Labels format manifest files. Multi-label classification is when an image is classified into a set of classes, but might belong to multiple classes at once. In this case, the image can potentially have multiple labels (multi-label), such as *football* and *ball*.

For information about multi-label SageMaker Ground truth jobs, see [Image Classification \(Multi-label\)](#). For information about multi-label format Amazon Rekognition Custom Labels manifest files, see [the section called "Adding multiple image-level labels to an image" \(p. 251\)](#).

Getting the manifest file for a SageMaker Ground Truth job

The following procedure shows you how to get the output manifest file (`output.manifest`) for an Amazon SageMaker Ground Truth job. You use `output.manifest` as input to the next procedure.

To download a SageMaker Ground Truth job manifest file

1. Open the <https://console.aws.amazon.com/sagemaker/>.
2. In the navigation pane, choose **Ground Truth** and then choose **Labeling Jobs**.
3. Choose the labeling job that contains the manifest file that you want to use.
4. On the details page, choose the link under **Output dataset location**. The Amazon S3 console is opened at the dataset location.
5. Choose **Manifests**, `output` and then `output.manifest`.
6. Choose **Object Actions** and then choose **Download** to download the manifest file.

Transforming a multi-label SageMaker manifest file

The following procedure creates a multi-label format Amazon Rekognition Custom Labels manifest file from an existing multi-label format SageMaker GroundTruth manifest file.

Note

To run the code, you need Python version 3, or higher.

To transform a multi-label SageMaker manifest file

1. Run the following python code. Supply the name of the manifest file that you created in [Getting the manifest file for a SageMaker Ground Truth job \(p. 266\)](#) as a command line argument.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0
"""
Purpose
Shows how to create an Amazon Rekognition Custom Labels format
manifest file from an Amazon SageMaker Ground Truth Image
Classification (Multi-label) format manifest file.
"""
import json
import logging
import argparse
import os.path

logger = logging.getLogger(__name__)

def create_manifest_file(ground_truth_manifest_file):
    """
    Creates an Amazon Rekognition Custom Labels format manifest file from
    an Amazon SageMaker Ground Truth Image Classification (Multi-label) format
    manifest file.
    :param ground_truth_manifest_file: The name of the Ground Truth manifest file,
    including the relative path.
    :return: The name of the new Custom Labels manifest file.
    """

    logger.info('Creating manifest file from %s', ground_truth_manifest_file)
    new_manifest_file = f'custom_labels_{os.path.basename(ground_truth_manifest_file)}'
```

```
# Read the SageMaker Ground Truth manifest file into memory.
with open(ground_truth_manifest_file) as gt_file:
    lines = gt_file.readlines()

#Iterate through the lines one at a time to generate the
#new lines for the Custom Labels manifest file.
with open(new_manifest_file, 'w') as the_new_file:
    for line in lines:
        #job_name - The of the Amazon Sagemaker Ground Truth job.
        job_name = ''
        # Load in the old json item from the Ground Truth manifest file
        old_json = json.loads(line)

        # Get the job name
        keys = old_json.keys()
        for key in keys:
            if 'source-ref' not in key and '-metadata' not in key:
                job_name = key

        new_json = {}
        # Set the location of the image
        new_json['source-ref'] = old_json['source-ref']

        # Temporarily store the list of labels
        labels = old_json[job_name]

        # Iterate through the labels and reformat to Custom Labels format
        for index, label in enumerate(labels):
            new_json[f'{job_name}{index}'] = index
            metadata = {}
            metadata['class-name'] = old_json[f'{job_name}-metadata']['class-map'][str(label)]
            metadata['confidence'] = old_json[f'{job_name}-metadata']['confidence-map'][str(label)]
            metadata['type'] = 'groundtruth/image-classification'
            metadata['job-name'] = old_json[f'{job_name}-metadata']['job-name']
            metadata['human-annotated'] = old_json[f'{job_name}-metadata']['human-annotated']
            metadata['creation-date'] = old_json[f'{job_name}-metadata']['creation-date']
            # Add the metadata to new json line
            new_json[f'{job_name}{index}-metadata'] = metadata
        # Write the current line to the json file
        the_new_file.write(json.dumps(new_json))
        the_new_file.write('\n')

    logger.info('Created %s', new_manifest_file)
    return new_manifest_file

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """
    parser.add_argument(
        "manifest_file", help="The Amazon SageMaker Ground Truth manifest file"
        "that you want to use."
    )

def main():
    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")
    try:
        # get command line arguments
```

```
parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
add_arguments(parser)
args = parser.parse_args()
# Create the manifest file
manifest_file = create_manifest_file(args.manifest_file)
print(f'Manifest file created: {manifest_file}')
except FileNotFoundError as err:
    logger.exception('File not found: %s', err)
    print(f'File not found: {err}. Check your manifest file.')

if __name__ == "__main__":
    main()
```

2. Note the name of the new manifest file that the script displays. You use it in the next step.
3. Do [Creating a manifest file \(p. 242\)](#) to create an Amazon Rekognition Custom Labels dataset with the new manifest file.

Note

You don't need to do step 6 of [Creating a manifest file \(p. 242\)](#) as the images are already uploaded to an Amazon S3 bucket. Make sure Amazon Rekognition Custom Labels has access to the Amazon S3 bucket referenced in the source-ref field of the manifest file JSON lines. For more information, see [Accessing external Amazon S3 Buckets \(p. 7\)](#). If your Ground Truth job stores images in the Amazon Rekognition Custom Labels Console Bucket, you don't need to add permissions.

Managing an Amazon Rekognition Custom Labels model

An Amazon Rekognition Custom Labels model is a mathematical model that predicts the presence of objects, scenes, and concepts in new images. It does this by finding patterns in images used to train the model. This section shows you how to train a model, evaluate its performance, and make improvements. It also shows you how to make a model available for use and how to delete a model when you no longer need it.

Topics

- [Deleting an Amazon Rekognition Custom Labels model \(p. 268\)](#)
- [Tagging a model \(p. 274\)](#)
- [Describing a model \(SDK\) \(p. 279\)](#)
- [Copying an Amazon Rekognition Custom Labels model \(SDK\) \(p. 284\)](#)

Deleting an Amazon Rekognition Custom Labels model

You can delete a model by using the Amazon Rekognition Custom Labels console or by using the [DeleteProjectVersion](#) API. You can't delete a model if it is running or if it is training. To stop a running model, use the [StopProjectVersion](#) API. For more information, see [Stopping an Amazon Rekognition Custom Labels model \(SDK\) \(p. 176\)](#). If a model is training, wait until it finishes before you delete the model.

A deleted model can't be undeleted.

Topics

- [Deleting an Amazon Rekognition Custom Labels model \(Console\) \(p. 269\)](#)

- [Deleting an Amazon Rekognition Custom Labels model \(SDK\) \(p. 269\)](#)

Deleting an Amazon Rekognition Custom Labels model (Console)

The following procedure shows how to delete a model from a project details page. You can also delete a model from a model's detail page.

To delete a model (console)

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. Choose **Use Custom Labels**.
3. Choose **Get started**.
4. In the left navigation pane, choose **Projects**.
5. Choose the project that contains the model that you want to delete. The project details page opens.
6. In the **Models** section, select the models that you want to delete.

Note

If the model can't be selected, the model is either running or is training, and can't be deleted. Check the **Status** field and try again after stopping the running model, or wait until training finishes.

7. Choose **Delete model** and the **Delete model dialog box is shown**.
8. Enter **delete** to confirm deletion.
9. Choose **Delete** to delete the model. Deleting the model might take a while to complete.

Note

If you **Close** the dialog box during model deletion, the models are still deleted.

Deleting an Amazon Rekognition Custom Labels model (SDK)

You delete an Amazon Rekognition Custom Labels model by calling [DeleteProjectVersion](#) and supplying the Amazon Resource Name (ARN) of the model that you want to delete. You can get the model ARN from the **Use your model** section of the model details page in the Amazon Rekognition Custom Labels console. Alternatively, call [DescribeProjectVersions](#) and supply the following.

- The ARN of the project (**ProjectArn**) that the model is associated with.
- The version name (**VersionNames**) of the model.

The model ARN is the **ProjectVersionArn** field in the **ProjectVersionDescription** object, from the [DescribeProjectVersions](#) response.

You can't delete a model if it is running or is training. To determine if the model is running or training, call [DescribeProjectVersions](#) and check the **Status** field of the model's **ProjectVersionDescription** object. To stop a running model, use the [StopProjectVersion](#) API. For more information, see [Stopping an Amazon Rekognition Custom Labels model \(SDK\) \(p. 176\)](#). You have to wait for a model to finish training before you can delete it.

To delete a model (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with **AmazonRekognitionFullAccess** permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).

- b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following code to delete a model.

AWS CLI

Change the value of `project-version-arn` to the name of the project that you want to delete.

```
aws rekognition delete-project-version --project-version-arn model_arn
```

Python

Supply the following command line parameters

- `project_arn` – the ARN of the project that contains the model that you want to delete.
- `model_arn` – the ARN of the model version that you want to delete.

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

"""
Purpose
Amazon Rekognition Custom Labels model example used in the service documentation:
https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/tm-delete-model.html
Shows how to delete an existing Amazon Rekognition Custom Labels model.
"""

import boto3
import argparse
import logging
import time
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def find_forward_slash(input_string, n):
    """
    Returns the location of '/' after n number of occurrences.
    :param input_string: The string you want to search
    :param n: the occurrence that you want to find.
    """
    position = input_string.find('/')
    while position >= 0 and n > 1:
        position = input_string.find('/', position + 1)
        n -= 1
    return position

def delete_model(rek_client, project_arn, model_arn):
    """
    Deletes an Amazon Rekognition Custom Labels model.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param model_arn: The ARN of the model version that you want to delete.
    """

    try:
        #Delete the model
        logger.info(f"Deleting dataset: {model_arn}")

        rek_client.delete_project_version(ProjectVersionArn=model_arn)

    except ClientError as e:
        logger.error(f"Error deleting dataset: {model_arn}. Error: {e}")

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("project_arn", help="The ARN of the project that contains the model to delete")
    parser.add_argument("model_arn", help="The ARN of the model version to delete")
    args = parser.parse_args()

    delete_model(rek_client, args.project_arn, args.model_arn)
```

```
# Get the model version name
start=find_forward_slash(model_arn,3) +1
end=find_forward_slash(model_arn,4)
version_name=model_arn[start:end]

deleted=False

#model might not be deleted yet, so wait deletion finishes.
while deleted==False:

    describe_response=rek_client.describe_project_versions(ProjectArn=project_arn,
        VersionNames=[version_name])
    if len(describe_response['ProjectVersionDescriptions']) == 0:
        deleted =True
    else:
        logger.info(f"Waiting for model deletion {model_arn}")
        time.sleep(5)

    logger.info(f"model deleted: {model_arn}")

return True

except ClientError as err:
    logger.exception(f"Couldn't delete model - {model_arn}:
{err.response['Error']['Message']}")

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project that contains the model that
you want to delete."
    )

    parser.add_argument(
        "model_arn", help="The ARN of the model version that you want to delete."
    )

def confirm_model_deletion(model_arn):
    """
    Confirms deletion of the model. Returns True if delete entered.
    :param model_arn: The ARN of the model that you want to delete.
    """

    print(f"Are you sure you want to delete model {model_arn} ?\n", model_arn)

    start = input("Enter delete to delete your model: ")
    if start == "delete":
        return True
    else:
        return False

def main():

    logging.basicConfig(level=logging.INFO, format"%(levelname)s: %(message)s")

    try:

        #get command line arguments
```

```
parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
add_arguments(parser)
args = parser.parse_args()

if confirm_model_deletion(args.model_arn)==True:
    print(f"Deleting model: {args.model_arn}")

    #Delete the model
    rek_client=boto3.client('rekognition')

    delete_model(rek_client,
                 args.project_arn,
                 args.model_arn)

    print(f"Finished deleting model: {args.model_arn}")
else:
    print(f"Not deleting model {args.model_arn}")

except ClientError as err:
    logger.exception(f"Problem deleting model: {err}")
    print(f"Problem deleting model: {err}")

if __name__ == "__main__":
    main()
```

Java 2

- `project_arn` – the ARN of the project that contains the model that you want to delete.
- `model_arn` – the ARN of the model version that you want to delete.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;

import software.amazon.awssdk.services.rekognition.RekognitionClient;

import
    software.amazon.awssdk.services.rekognition.model.DeleteProjectVersionRequest;
import
    software.amazon.awssdk.services.rekognition.model.DeleteProjectVersionResponse;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsResponse;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

public class DeleteModel {

    public static final Logger logger =
    Logger.getLogger(DeleteModel.class.getName());

    public static int findForwardSlash(String modelArn, int n) {

        int start = modelArn.indexOf('/');
        while (start >= 0 && n > 1) {
            start = modelArn.indexOf('/', start + 1);
        }
    }
}
```

```
        n -= 1;
    }
    return start;
}

public static void deleteMyModel(RekognitionClient rekClient, String
projectArn, String modelArn)
    throws InterruptedException {

    try {

        logger.log(Level.INFO, "Deleting model: {0}", projectArn);

        // Delete the model

        DeleteProjectVersionRequest deleteProjectVersionRequest =
DeleteProjectVersionRequest.builder()
            .projectVersionArn(modelArn).build();

        DeleteProjectVersionResponse response =
            rekClient.deleteProjectVersion(deleteProjectVersionRequest);

        logger.log(Level.INFO, "Status: {0}", response.status());

        // Get the model version

        int start = findForwardSlash(modelArn, 3) + 1;
        int end = findForwardSlash(modelArn, 4);

        String versionName = modelArn.substring(start, end);

        Boolean deleted = false;

        DescribeProjectVersionsRequest describeProjectVersionsRequest =
DescribeProjectVersionsRequest.builder()
            .projectArn(projectArn).versionNames(versionName).build();

        // Wait until model is deleted.

        do {

            DescribeProjectVersionsResponse describeProjectVersionsResponse =
rekClient
            .describeProjectVersions(describeProjectVersionsRequest);

            if
(describeProjectVersionsResponse.projectVersionDescriptions().size()==0) {
                logger.log(Level.INFO, "Waiting for model deletion: {0}",
modelArn);
                Thread.sleep(5000);
            } else {
                deleted = true;
                logger.log(Level.INFO, "Model deleted: {0}", modelArn);
            }
        } while (Boolean.FALSE.equals(deleted));

        logger.log(Level.INFO, "Model deleted: {0}", modelArn);

    } catch (
RekognitionException e) {
    logger.log(Level.SEVERE, "Client error occurred: {0}", e.getMessage());
    throw e;
}
}
```

```
    }

    public static void main(String args[]) {

        final String USAGE = "\n" + "Usage: " + "<project_arn> <model_arn>\n\n" +
"Where:\n" +
        "    + " project_arn - The ARN of the project that contains the model
        that you want to delete.\n\n" +
        "    + " model_version - The ARN of the model that you want to delete.
\n\n";

        if (args.length != 2) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String projectArn = args[0];
        String modelVersion = args[1];

        try {

            RekognitionClient rekClient = RekognitionClient.builder().build();

            // Delete the model
            deleteMyModel(rekClient, projectArn, modelVersion);

            System.out.println(String.format("model deleted: %s", modelVersion));

            rekClient.close();

        } catch (RekognitionException rekError) {
            logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
            System.exit(1);
        }

        catch (InterruptedException intError) {
            logger.log(Level.SEVERE, "Exception while sleeping: {0}",
intError.getMessage());
            System.exit(1);
        }

    }

}
```

Tagging a model

You can identify, organize, search for, and filter your Amazon Rekognition models by using tags. Each tag is a label consisting of a user-defined key and value. For example, to help determine billing for your models, tag your models with a `Cost_center` key and add the appropriate cost center number as a value. For more information, see [Tagging AWS resources](#).

Use tags to:

- Track billing for a model by using cost allocation tags. For more information, see [Using Cost Allocation Tags](#).
- Control access to a model by using Identity and Access Management (IAM). For more information, see [Controlling access to AWS resources using resource tags](#).

- Automate model management. For example, you can run automated start or stop scripts that turn off development models during non-business hours to reduce costs. For more information, see [Running a trained Amazon Rekognition Custom Labels model \(p. 165\)](#).

You can tag models by using the Amazon Rekognition console or by using the AWS SDKs.

Topics

- [Tagging models \(console\) \(p. 275\)](#)
- [Viewing model tags \(p. 275\)](#)
- [Tagging models \(SDK\) \(p. 276\)](#)

Tagging models (console)

You can use the Rekognition console to add tags to models, view the tags attached to a model, and remove tags.

Adding or removing tags

This procedure explains how to add tags to, or remove tags from, an existing model. You can also add tags to a new model when it is trained. For more information, see [Training an Amazon Rekognition Custom Labels model \(p. 106\)](#).

To add tags to, or remove tags from, an existing model using the console

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.
2. Choose **Get started**.
3. In the navigation pane, choose **Projects**.
4. On the **Projects** resources page, choose the project that contains the model that you want to tag.
5. In the navigation pane, under the project you previously chose, choose **Models**.
6. In the **Models** section, choose the model that you want to add a tag to.
7. On the model's details page, choose the **Tags** tab.
8. In the **Tags** section, choose **Manage tags**.
9. On the **Manage tags** page, choose **Add new tag**.
10. Enter a key and a value.
 - a. For **Key**, enter a name for the key.
 - b. For **Value**, enter a value.
11. To add more tags, repeat steps 9 and 10.
12. (Optional) To remove a tag, choose **Remove** next to the tag that you want to remove. If you are removing a previously saved tag, it is removed when you save your changes.
13. Choose **Save changes** to save your changes.

Viewing model tags

You can use the Amazon Rekognition console to view the tags attached to a model.

To view the tags attached to *all models within a project*, you must use the AWS SDK. For more information, see [Listing model tags \(p. 277\)](#).

To view the tags attached to a model

1. Open the Amazon Rekognition console at <https://console.aws.amazon.com/rekognition/>.

2. Choose **Get started**.
3. In the navigation pane, choose **Projects**.
4. On the **Projects** resources page, choose the project that contains the model whose tag you want to view.
5. In the navigation pane, under the project you previously chose, choose **Models**.
6. In the **Models** section, choose the model whose tag you want to view.
7. On the model's details page, choose the **Tags** tab. The tags are shown in **Tags** section.

Tagging models (SDK)

You can use the AWS SDK to:

- Add tags to a new model
- Add tags to an existing model
- List the tags attached to a model
- Remove tags from a model

The tags in the following AWS CLI examples are in the following format.

```
--tags '{"key1":"value1","key2":"value2"}'
```

Alternatively, you can use this format.

```
--tags key1=value1,key2=value2
```

If you haven't installed the AWS CLI, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).

Adding tags to a new model

You can add tags to a model when you create it using the [CreateProjectVersion](#) operation. Specify one or more tags in the Tags array input parameter.

```
aws rekognition create-project-version --project-name "project name" \
--output-config '{ "S3Location": { "Bucket": "output bucket", "Prefix": "output
folder" } }' \
--tags '{"key1":"value1","key2":"value2"}'
```

For information about creating and training a model, see [Training a model \(SDK\) \(p. 109\)](#).

Adding tags to an existing model

To add one or more tags to an existing model, use the [TagResource](#) operation. Specify the model's Amazon Resource Name (ARN) (ResourceArn) and the tags (Tags) that you want to add. The following example shows how to add two tags.

```
aws rekognition tag-resource --resource-arn resource-arn \
--tags '{"key1":"value1","key2":"value2"}'
```

You can get the ARN for a model by calling [CreateProjectVersion](#).

Listing model tags

To list the tags attached to a model, use the [ListTagsForResource](#) operation and specify the ARN of the model (`ResourceArn`). The response is a map of tag keys and values that are attached to the specified model.

```
aws rekognition list-tags-for-resource --resource-arn resource-arn
```

The output displays a list of the tags attached to the model.

```
{  
  "Tags": [  
    {"Dept": "Engineering",  
     "Name": "Ana Silva Carolina",  
     "Role": "Developer"}  
  ]  
}
```

To see which models in a project have a specific tag, call `DescribeProjectVersions` to get a list of models. Then call `ListTagsForResource` for each model in the response from `DescribeProjectVersions`. Inspect the response from `ListTagsForResource` to see if the required tag is present.

The following Python 3 example shows you how search all your projects for a specific tag key and value. The output includes the project ARN and the model ARN where a matching key is found.

To search for a tag value

1. Save the following code to a file named `find_tag.py`.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
"""  
Purpose  
Shows how to find a tag value that's associated with models within  
your Amazon Rekognition Custom Labels projects.  
To run: python find_tag.py tag tag_value  
"""  
import logging  
import argparse  
import boto3  
  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)  
  
def find_tag_in_projects(rekognition_client, key, value):  
    """  
    Finds Amazon Rekognition Custom Label models tagged with the supplied key and key  
    value.  
    :param rekognition_client: An Amazon Rekognition boto3 client.  
    :param key: The tag key to find.  
    :param value: The value of the tag that you want to find.  
    return: A list of matching model versions (and model projects) that were found.  
    """  
    try:  
        found_tags = []
```

```
found = False

projects = rekognition_client.describe_projects()
# Iterate through each project and models within a project.
for project in projects["ProjectDescriptions"]:
    logger.info("Searching project: %s ...", project["ProjectArn"])

    models = rekognition_client.describe_project_versions(
        ProjectArn=(project["ProjectArn"]))
    )

    for model in models["ProjectVersionDescriptions"]:
        logger.info("Searching model %s", model["ProjectVersionArn"])

        tags = rekognition_client.list_tags_for_resource(
            ResourceArn=model["ProjectVersionArn"])
        )

        logger.info(
            "\t\tSearching model: %s for tag: %s value: %s.",
            model["ProjectVersionArn"],
            key,
            value,
        )
        # Check if tag exists

        if key in tags["Tags"]:
            if tags["Tags"][key] == value:
                found = True
                logger.info(
                    "\t\t\tMATCH: Project: %s: model version %s",
                    project["ProjectArn"],
                    model["ProjectVersionArn"],
                )
                found_tags.append(
                    {
                        "Project": project["ProjectArn"],
                        "ModelVersion": model["ProjectVersionArn"],
                    }
                )
            )

        if found is False:
            logger.info("No match for Tag %s with value %s.", key, value)
    return found_tags
except ClientError as err:
    logger.info("Problem finding tags: %s. ", format(err))
    raise

def main():
    """
    Entry point for example.
    """
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    # Set up command line arguments.
    parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)

    parser.add_argument("tag", help="The tag that you want to find.")
    parser.add_argument("value", help="The tag value that you want to find.")

    args = parser.parse_args()
    key = args.tag
    value = args.value

    print(
```

```
        "Searching your models for tag: {tag} with value: {value}.".format(
            tag=key, value=value
        )
    )

rekognition_client = boto3.client("rekognition")

# Get tagged models for all projects.
tagged_models = find_tag_in_projects(rekognition_client, key, value)

print("Matched models\n-----")
if len(tagged_models) > 0:
    for model in tagged_models:
        print(
            "Project: {project}\nModel version: {version}\n".format(
                project=model["Project"], version=model["ModelVersion"]
            )
        )
else:
    print("No matches found.")

print("Done.")

if __name__ == "__main__":
    main()
```

2. At the command prompt, enter the following. Replace **key** and **value** with the key name and the key value that you want to find.

```
python find_tag.py key value
```

Deleting tags from a model

To remove one or more tags from a model, use the [UntagResource](#) operation. Specify the ARN of the model (ResourceArn) and the tag keys (Tag-Keys) that you want to remove.

```
aws rekognition untag-resource --resource-arn resource-arn \
--tag-keys '["key1", "key2"]'
```

Alternatively, you can specify tag-keys in this format.

```
--tag-keys key1, key2
```

Describing a model (SDK)

You can use the [DescribeProjectVersions](#) API to get information about a version of a model. If you don't specify VersionName, [DescribeProjectVersions](#) returns descriptions for all model versions in the project.

To describe a model (SDK)

1. If you haven't already:

- a. Create or update an IAM user with AmazonRekognitionFullAccess permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).

- b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following example code to describe a version of a model.

AWS CLI

Change the value of `project-arn` to the ARN of the project that you want to describe. Change the value of `version-name` to the version of the model that you want to describe.

```
aws rekognition describe-project-versions --project-arn project_arn \  
--version-names version_name
```

Python

Use the following code. Supply the following command line parameters:

- `project_arn` — the ARN of the model that you want to describe.
- `model_version` — the version of the model that you want to describe.

For example: `python describe_model.py project_arn model_version`

```
#Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.  
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)  
  
import boto3  
import argparse  
import logging  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)  
  
def describe_model(rek_client, project_arn, version_name):  
    """  
    Describes an Amazon Rekognition Custom Labels model.  
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.  
    :param dataset_arn: The ARN of the model that you want to describe.  
    """  
  
    try:  
        #Describe the model  
        logger.info(f"Describing model: {version_name} for project {project_arn}")  
  
        describe_response=rek_client.describe_project_versions(ProjectArn=project_arn,  
                                                               VersionNames=[version_name])  
        for model in describe_response['ProjectVersionDescriptions']:  
            print(f"Created: {str(model['CreationTimestamp'])} ")  
            print(f"ARN: {str(model['ProjectVersionArn'])} ")  
            if 'BillableTrainingTimeInSeconds' in model:  
                print(f"Billing training time (minutes):  
{str(model['BillableTrainingTimeInSeconds']/60)} ")  
                print("Evaluation results: ")  
                if 'EvaluationResult' in model:  
                    evaluation_results = model['EvaluationResult']  
                    print(f"\tF1 score: {str(evaluation_results['F1Score'])}")  
                    print(f"\tSummary location: s3://{evaluation_results['Summary']}  
['S3Object']['Bucket']}/{evaluation_results['Summary']['S3Object']['Name']}")  
                if 'ManifestSummary' in model:
```

```
        print(f"Manifest summary location: s3://{model['ManifestSummary']}"
['S3Object']['Bucket']}/{model['ManifestSummary']['S3Object']['Name']}")
        if 'OutputConfig' in model:
            print(f"Training output location: s3://{model['OutputConfig']}"
['S3Bucket']}/{model['OutputConfig']['S3KeyPrefix']}")
            if 'MinInferenceUnits' in model:
                print(f"Minimum inference units:
{str(model['MinInferenceUnits'])}")
            if 'MaxInferenceUnits' in model:
                print(f"Maximum Inference units:
{str(model['MaxInferenceUnits'])}")

            print("Status: " + model['Status'])
            print("Message: " + model['StatusMessage'])

    except ClientError as err:
        logger.exception(f"Couldn't describe model: {err.response['Error']}"
['Message']")
        raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The ARN of the project in which the model resides."
    )
    parser.add_argument(
        "version_name", help="The version of the model that you want to describe."
    )

def main():
    logging.basicConfig(level=logging.INFO, format"%(levelname)s: %(message)s")
    try:
        #get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        print(f"Describing model: {args.version_name} for project
{args.project_arn}.")

        #Describe the model
        rek_client=boto3.client('rekognition')

        describe_model(rek_client, args.project_arn,
                      args.version_name
        )

        print(f"Finished describing model: {args.version_name} for project
{args.project_arn}.")

    except ClientError as err:
        logger.exception(f"Problem describing model: {err}")
        print(f"Problem describing model: {err}")
    except Exception as err:
```

```
logger.exception(f"Problem describing model: {err}")
print(f"Problem describing model: {err}")

if __name__ == "__main__":
    main()
```

Java 2

Use the following code. Supply the following command line parameters:

- `project_arn` — the ARN of the model that you want to describe.
- `model_version` — the version of the model that you want to describe.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsRequest;
import
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsResponse;
import software.amazon.awssdk.services.rekognition.model.EvaluationResult;
import software.amazon.awssdk.services.rekognition.model.GroundTruthManifest;
import software.amazon.awssdk.services.rekognition.model.OutputConfig;
import software.amazon.awssdk.services.rekognition.model.ProjectVersionDescription;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

import java.net.URI;
import java.util.logging.Level;
import java.util.logging.Logger;

public class DescribeModel {

    public static final Logger logger =
Logger.getLogger(DescribeModel.class.getName());

    public static void describeMyModel(RekognitionClient rekClient, String
projectArn, String versionName) {

        try {

            // If a single version name is supplied, build request argument
            DescribeProjectVersionsRequest describeProjectVersionsRequest = null;

            if (versionName == null) {
                describeProjectVersionsRequest =
DescribeProjectVersionsRequest.builder().projectArn(projectArn)
                    .build();
            } else {
                describeProjectVersionsRequest =
DescribeProjectVersionsRequest.builder().projectArn(projectArn)
                    .versionNames(versionName).build();
            }

            DescribeProjectVersionsResponse describeProjectVersionsResponse =
rekClient
                .describeProjectVersions(describeProjectVersionsRequest);
```

```
        for (ProjectVersionDescription projectVersionDescription :  
describeProjectVersionsResponse  
        .projectVersionDescriptions()) {  
  
            System.out.println("ARN: " +  
projectVersionDescription.projectVersionArn());  
            System.out.println("Status: " +  
projectVersionDescription.statusAsString());  
            System.out.println("Message: " +  
projectVersionDescription.statusMessage());  
  
            if (projectVersionDescription.billableTrainingTimeInSeconds() !=  
null) {  
                System.out.println(  
                    "Billable minutes: " +  
(projectVersionDescription.billableTrainingTimeInSeconds() / 60));  
            }  
  
            if (projectVersionDescription.evaluationResult() != null) {  
                EvaluationResult evaluationResult =  
projectVersionDescription.evaluationResult();  
  
                System.out.println("F1 Score: " + evaluationResult.f1Score());  
                System.out.println("Summary location: s3://" +  
evaluationResult.summary().s3Object().bucket() + "/"  
                    + evaluationResult.summary().s3Object().name());  
            }  
  
            if (projectVersionDescription.manifestSummary() != null) {  
                GroundTruthManifest manifestSummary =  
projectVersionDescription.manifestSummary();  
                System.out.println("Manifest summary location: s3://" +  
manifestSummary.s3Object().bucket() + "/"  
                    + manifestSummary.s3Object().name());  
            }  
  
            if (projectVersionDescription.outputConfig() != null) {  
                OutputConfig outputConfig =  
projectVersionDescription.outputConfig();  
                System.out.println(  
                    "Training output: s3://" + outputConfig.s3Bucket() +  
"/" + outputConfig.s3KeyPrefix());  
            }  
  
            if (projectVersionDescription.minInferenceUnits() != null) {  
                System.out.println("Min inference units: " +  
projectVersionDescription.minInferenceUnits());  
            }  
  
            System.out.println();  
        }  
  
    } catch (RekognitionException rekError) {  
        logger.log(Level.SEVERE, "Rekognition client error: {0}",  
rekError.getMessage());  
        throw rekError;  
    }  
}  
  
public static void main(String args[]) {  
  
    String projectArn = null;  
    String versionName = null;
```

```
final String USAGE = "\n" + "Usage: " + "<project_arn> <version_name>\n\n"
+ "Where:\n"
+ "    + " project_arn - The ARN of the project that contains the models
you want to describe.\n\n"
+ "    + " version_name - (optional) The version name of the model that
you want to describe. \n\n"
+ "                If you don't specify a value, all model
versions are described.\n\n";

if (args.length > 2 || args.length == 0) {
    System.out.println(USAGE);
    System.exit(1);
}

projectArn = args[0];

if (args.length == 2) {
    versionName = args[1];
}

try {

    // Get the Rekognition client
    RekognitionClient rekClient = RekognitionClient.builder().build();

    // Describe the model
    describeMyModel(rekClient, projectArn, versionName);

    rekClient.close();

} catch (RekognitionException rekError) {
    logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
    System.exit(1);
}

}
```

Copying an Amazon Rekognition Custom Labels model (SDK)

You can use the [CopyProjectVersion](#) operation to copy an Amazon Rekognition Custom Labels model version from a source Amazon Rekognition Custom Labels project to a destination project. The destination project can be in a different AWS account, or in the same AWS account. A typical scenario is copying a tested model from a development AWS account to a production AWS account.

Alternatively, you can train the model in the destination account with the source dataset. Using the [CopyProjectVersion](#) operation has the following advantages.

- Model behavior is consistent. Model training is non-deterministic and two models trained with same dataset aren't guaranteed to make the same predictions. Copying the model with [CopyProjectVersion](#) helps make sure that the behavior of the copied model is consistent with the source model and you won't need to re-test the model.
- Model training isn't required. This saves you money as you are charged for each successful training of a model.

To copy a model to a different AWS account, you must have an Amazon Rekognition Custom Labels project in the destination AWS account. For information about creating a project, see [Creating a project \(p. 50\)](#). Be sure to create the project in the destination AWS account.

A [project policy \(p. 285\)](#) is a resource-based policy that sets copy permissions for the model version that you want to copy. You will need to use a [project policy \(p. 285\)](#) when the destination project is in a different AWS account from the source project.

You do not need to use a [project policy \(p. 285\)](#), when copying model versions within the same account. However, you can choose to use a [project policy \(p. 285\)](#) on inter-account projects if you would like more control over these resources.

You attach the project policy to the source project by calling the [PutProjectPolicy](#) operation.

You can't use [CopyProjectVersion](#) to copy a model to a project in a different AWS Region. Also, you can't copy a model with the Amazon Rekognition Custom Labels console. In these cases, you can train the model in the destination project with the datasets used to train the source model. For more information, see [Training an Amazon Rekognition Custom Labels model \(p. 106\)](#).

To copy a model from a source project to a destination project, do the following:

To copy a model

1. [Create a project policy document \(p. 285\)](#).
2. [Attach the project policy to the source project \(p. 287\)](#).
3. [Copy the model with the CopyProjectVersion operation \(p. 292\)](#).

To remove a project policy from a project, call [DeleteProjectPolicy](#). To get a list of project policies attached to a project, call [ListProjectPolicies](#).

Topics

- [Creating a project policy document \(p. 285\)](#)
- [Attaching a project policy \(SDK\) \(p. 287\)](#)
- [Copying a model \(SDK\) \(p. 292\)](#)
- [Listing project policies \(SDK\) \(p. 300\)](#)
- [Deleting a project policy \(SDK\) \(p. 304\)](#)

Creating a project policy document

Rekognition Custom Labels uses a resource-based policy, known as *project policy*, to manage copy permissions for a model version. A project policy is a JSON format document.

A project policy allows or denies a [principal](#) permission to copy a model version from a source project to a destination project. You need a project policy if the destination project is in a different AWS account. That's also true if the destination project is in the same AWS account as the source project and you want to restrict access to specific model versions. For example, you might want to deny copy permissions to a specific IAM role within an AWS account.

The following example allows the principal `arn:aws:iam::111111111111:role/Admin` to copy the model version `arn:aws:rekognition:us-east-1:123456789012:project/my_project/version/test_1/1627045542080`.

```
{  
  "Version": "2012-10-17",
```

```
"Statement": [
  {
    "Effect": "Allow",
    "Principal": {
      "AWS": "arn:aws:iam::111111111111:role/Admin"
    },
    "Action": "rekognition:CopyProjectVersion",
    "Resource": "arn:aws:rekognition:us-east-1:111111111111:project/my_project/version/
test_1/1627045542080"
  }
]
```

Note

Action, Resource, Principal, and Effect are required fields in a project policy document.

The only supported action is rekognition:CopyProjectVersion.

NotAction, NotResource, and NotPrincipal are prohibited fields and must not be present in the project policy document.

If you don't specify a project policy, a principal in the same AWS account as the source project can still copy a model, if the principal has an Identity-based policy, such as AmazonRekognitionCustomLabelsFullAccess, that gives permission to call CopyProjectVersion.

The following procedure creates a project policy document file that you can use with the Python example in [Attaching a project policy \(SDK\) \(p. 287\)](#). If you are using the put-project-policy AWS CLI command, you supply the project policy as a JSON string.

To create a project policy document

1. In a text editor, create the following document. Change the following values:
 - Effect – Specify ALLOW to grant copy permission. Specify DENY to deny copy permission.
 - Principal – To the principal that you want to allow or deny access to the model versions that you specify in Resource. For example you can specify the [AWS account principal](#) for a different AWS account. We don't restrict the principals that you can use. For more information, see [Specifying a principal](#).
 - Resource – The Amazon Resource Name (ARN) of the model version for which you want to specify copy permissions. If you want to grant permissions to all model versions within the source project, use the following format `arn:aws:rekognition:region:account:project/source project/version/*`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "ALLOW or DENY",
      "Principal": {
        "AWS": "principal"
      },
      "Action": "rekognition:CopyProjectVersion",
      "Resource": "Model version ARN"
    }
  ]
}
```

2. Save the project policy to your computer.
3. Attach the project policy to the source project by following the instructions at [Attaching a project policy \(SDK\) \(p. 287\)](#).

Attaching a project policy (SDK)

You attach a project policy to an Amazon Rekognition Custom Labels project by calling the [PutProjectPolicy](#) operation.

Attach multiple project policies to a project by calling `PutProjectPolicy` for each project policy that you want to add. You can attach up to five project project policies to a project. If you need to attach more project policies, you can request a [limit \(p. 323\)](#) increase.

When you first attach a unique project policy to a project, don't specify a revision ID in the `PolicyRevisionId` input parameter. The response from `PutProjectPolicy` is a revision ID for the project policy that Amazon Rekognition Custom Labels creates for you. You can use the revision ID to update or delete the latest revision of a project policy. Amazon Rekognition Custom Labels only keeps the latest revision of a project policy. If you try to update or delete a previous revision of a project policy, you get an `InvalidPolicyRevisionIdException` error.

To update an existing project policy, specify the revision ID of the project policy in the `PolicyRevisionId` input parameter. You can get the revision IDs for project policies in a project by calling [ListProjectPolicies](#).

After you attach a project policy to a source project, you can copy the model from the source project to the destination project. For more information, see [Copying a model \(SDK\) \(p. 292\)](#).

To remove a project policy from a project, call [DeleteProjectPolicy](#). To get a list of project policies attached to a project, call [ListProjectPolicies](#).

To attach a project policy to a project (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with `AmazonRekognitionCustomLabelsFullAccess` permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
 - c. [Create a project policy document \(p. 285\)](#).
2. Use the following code to attach the project policy to the project, in the trusting AWS account, that contains the model version that you want to copy. To get the project ARN, call [DescribeProjects](#). To get the model version ARN call [DescribeProjectVersions](#).

AWS CLI

Change the following values:

- `project-arn` to the ARN of the source project in the trusting AWS account that contains the model version that you want to copy.
- `policy-name` to a policy name that you choose.
- `principal` To the principal that you want to allow or deny access to the model versions that you specify in `Model version ARN`.
- `project-version-arn` to the ARN of the model version that you want to copy.

If you want to update an existing project policy, specify the `policy-revision-id` parameter and supply the revision ID of the desired project policy.

```
aws rekognition put-project-policy \
--project-arn project-arn \
--policy-name policy-name \
```

```
--policy-document '{ "Version":"2012-10-17", "Statement":  
[ { "Effect":"ALLOW or DENY", "Principal":{ "AWS":"principal" },  
"Action":"rekognition:CopyProjectVersion", "Resource":"project-version-arn" }]}'
```

Python

Use the following code. Supply the following command line parameters:

- `project_arn` – The ARN of the source project that you want to attach the project policy to.
- `policy_name` – A policy name that you choose.
- `project_policy` – The file that contains the project policy document.
- `policy_revision_id` – (Optional). If you want to update an existing revision of a project policy, specify the revision ID of the project policy.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# SPDX-License-Identifier: Apache-2.0  
  
"""  
Purpose  
Amazon Rekognition Custom Labels model example used in the service documentation:  
https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/md-copy-model-sdk.html  
Shows how to attach a project policy to an Amazon Rekognition Custom Labels  
project.  
"""  
  
import boto3  
import argparse  
import logging  
import json  
from botocore.exceptions import ClientError  
  
logger = logging.getLogger(__name__)  
  
def put_project_policy(rek_client, project_arn, policy_name, policy_document_file,  
policy_revision_id=None):  
    """  
        Attaches a project policy to an Amazon Rekognition Custom Labels project.  
        :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.  
        :param policy_name: A name for the project policy.  
        :param project_arn: The Amazon Resource Name (ARN) of the source project that  
        you want to attach the project policy to.  
        :param policy_document_file: The JSON project policy document to attach to the  
        source project.  
        :param policy_revision_id: (Optional) The revision of an existing policy to  
        update. Pass None to attach new policy.  
        :return The revision ID for the project policy.  
    """  
  
    try:  
        policy_document_json = ""  
        response = None  
  
        with open(policy_document_file, 'r') as policy_document:  
            policy_document_json = json.dumps(json.load(policy_document))  
  
        logger.info(  
            f"Attaching {policy_name} project_policy to project {project_arn}.")
```

```
if policy_revision_id is None:
    response = rek_client.put_project_policy(ProjectArn=project_arn,
                                              PolicyName=policy_name,
                                              PolicyDocument=policy_document_json)

    else:
        response = rek_client.put_project_policy(ProjectArn=project_arn,
                                              PolicyName=policy_name,
                                              PolicyDocument=policy_document_json,
                                              PolicyRevisionId=policy_revision_id)

        new_revision_id = response['PolicyRevisionId']

        logger.info(
            f"Finished creating project policy {policy_name}. Revision ID: {new_revision_id}")

        return new_revision_id

    except ClientError as err:
        logger.exception(
            f"Couldn't attach {policy_name} project policy to project {project_arn}: {err.response['Error']['Message']}")
        raise

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project_arn", help="The Amazon Resource Name (ARN) of the project "
        "that you want to attach the project policy to."
    )
    parser.add_argument(
        "policy_name", help="A name for the project policy."
    )
    parser.add_argument(
        "project_policy", help="The file containing the project policy JSON"
    )
    parser.add_argument(
        "--policy_revision_id", help="The revision of an existing policy to
        update. "
        "If you don't supply a value, a new project policy is created.",
        required=False
    )

def main():
    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:
        # get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
```

```
add_arguments(parser)

args = parser.parse_args()

print(f"Attaching policy to {args.project_arn}")

rek_client = boto3.client('rekognition')

# Attach a new policy or update an existing policy.

response = put_project_policy(rek_client,
                                args.project_arn,
                                args.policy_name,
                                args.project_policy,
                                args.policy_revision_id)

print(
    f"project policy {args.policy_name} attached to project
{args.project_arn}")
print(f"Revision ID: {response}")

except ClientError as err:
    logger.exception(f"Problem attaching project policy: {err}")
    print(f"Problem attaching project policy: {err}")

if __name__ == "__main__":
    main()
```

Java 2

Use the following code. Supply the following command line parameters:

- **project_arn** – The ARN of the source project that you want to attach the project policy to.
- **project_policy_name** – A policy name that you choose.
- **project_policy_document** – The file that contains the project policy document.
- **project_policy_revision_id** – (Optional). If you want to update an existing revision of a project policy, specify the revision ID of the project policy.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import java.io.IOException;
import java.net.URI;
import java.net.URISyntaxException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.logging.Level;
import java.util.logging.Logger;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import software.amazon.awssdk.services.rekognition.model.PutProjectPolicyRequest;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

public class PutProjectPolicy {

    public static final Logger logger =
        Logger.getLogger(PutProjectPolicy.class.getName());
```

```
    public static void putMyProjectPolicy(RekognitionClient rekClient, String
projectArn, String projectPolicyName,
                                         String projectPolicyFileName, String projectPolicyRevisionId) throws
IOException {
    try {
        Path filePath = Path.of(projectPolicyFileName);

        String policyDocument = Files.readString(filePath);

        String[] logArguments = new String[] { projectPolicyFileName,
projectPolicyName };

        PutProjectPolicyRequest putProjectPolicyRequest = null;

        logger.log(Level.INFO, "Attaching Project policy: {0} to project: {1}",
logArguments);

        // Attach the project policy.

        if (projectPolicyRevisionId == null) {
            putProjectPolicyRequest =
PutProjectPolicyRequest.builder().projectArn(projectArn)

            .policyName(projectPolicyName).policyDocument(policyDocument).build();
        } else {
            putProjectPolicyRequest =
PutProjectPolicyRequest.builder().projectArn(projectArn)

            .policyName(projectPolicyName).policyRevisionId(projectPolicyRevisionId)
            .policyDocument(policyDocument)

            .build();
        }
        rekClient.putProjectPolicy(putProjectPolicyRequest);

        logger.log(Level.INFO, "Attached Project policy: {0} to project: {1}",
logArguments);
    } catch (
        RekognitionException e) {
        logger.log(Level.SEVERE, "Client error occurred: {0}", e.getMessage());
        throw e;
    }
}

public static void main(String args[]) {
    final String USAGE = "\n" + "Usage: "
        + "<project_arn> <project_policy_name> <policy_document>
<project_policy_revision_id>\n\n" + "Where:\n"
        + "    project_arn - The ARN of the project that you want to attach
the project policy to.\n\n"
        + "    project_policy_name - A name for the project policy.\n\n"
        + "    project_policy_document - The file name of the project
policy.\n\n"
        + "    project_policy_revision_id - (Optional) The revision ID of
the project policy that you want to update.\n\n";
    if (args.length < 3 || args.length > 4) {
        System.out.println(USAGE);
    }
}
```

```
        System.exit(1);
    }

    String projectArn = args[0];
    String projectPolicyName = args[1];
    String projectPolicyDocument = args[2];
    String projectPolicyRevisionId = null;

    if (args.length == 4) {
        projectPolicyRevisionId = args[3];
    }

    try {
        RekognitionClient rekClient = RekognitionClient.builder().build();

        // Attach the project policy.
        putMyProjectPolicy(rekClient, projectArn, projectPolicyName,
        projectPolicyDocument,
        projectPolicyRevisionId);

        System.out.println(
            String.format("project policy %s: attached to project: %s",
        projectPolicyName, projectArn));

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    }

    catch (IOException intError) {
        logger.log(Level.SEVERE, "Exception while reading policy document:
{0}", intError.getMessage());
        System.exit(1);
    }

}
}
```

3. Copy the model version by following the instructions at [Copying a model \(SDK\) \(p. 292\)](#).

Copying a model (SDK)

You can use the `CopyProjectVersion` API to copy a model version from a source project to a destination project. The destination project can be in a different AWS account but must be the same AWS Region. If the destination project is in a different AWS account (or if you want to grant specific permissions for a model version copied within an AWS account), you must attach a project policy to the source project. For more information, see [Creating a project policy document \(p. 285\)](#). The `CopyProjectVersion` API requires access to your Amazon S3 bucket.

The copied model includes the training results for the source model, but doesn't include the source datasets.

The source AWS account has no ownership over the model copied into a destination account, unless you set up appropriate permissions.

To copy a model (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with `AmazonRekognitionFullAccess` permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Attach a project policy to the source project by following the instructions at [Attaching a project policy \(SDK\) \(p. 287\)](#).
3. If you are copying the model to a different AWS account, make sure that you have a project in the destination AWS account.
4. Use the following code to copy the model version to a destination project.

AWS CLI

Change the following values:

- `source-project-arn` to the ARN of the source project that contains the model version that you want to copy.
- `source-project-version-arn` to the ARN of the model version that you want to copy.
- `destination-project-arn` to the ARN of the destination project that you want to copy the model to.
- `version-name` to a version name for the model in the destination project.
- `bucket` to the S3 bucket that you want the training results for the source model copied to.
- `folder` to the folder in `bucket` that you want the training results for the source model copied to.
- (Optional) `kms-key-id` to the AWS Key Management Service key ID for the model.
- (Optional) `key` to a tag key of your choosing.
- (Optional) `value` to a tag value of your choosing.

```
aws rekognition copy-project-version \
--source-project-arn source-project-arn \
--source-project-version-arn source-project-version-arn \
--destination-project-arn destination-project-arn \
--version-name version-name \
--output-config '{"S3Bucket":"'bucket'", "S3KeyPrefix":"'folder'"}' \
--kms-key-id arn:myKey \
--tags "{'key':'key'}"
```

Python

Use the following code. Supply the following command line parameters:

- `source_project_arn` — the ARN of the source project in the source AWS account that contains the model version that you want to copy.
- `source_project_version_arn` — the ARN of the model version in the source AWS account that you want to copy.
- `destination_project_arn` — the ARN of the destination project that you want to copy the model to.
- `destination_version_name` — a version name for the model in the destination project.

- `training_results` — the S3 location that you want the training results for the source model version copied to.
- (Optional) `kms_key_id` to the AWS Key Management Service key ID for the model.
- (Optional) `tag_name` to a tag key of your choosing.
- (Optional) `tag_value` to a tag value of your choosing.

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

import boto3
import argparse
import logging
import time
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def copy_model(
    rekognition_client, source_project_arn, source_project_version_arn,
    destination_project_arn, training_results, destination_version_name):
    """
    Copies a version of a Amazon Rekognition Custom Labels model.

    :param rekognition_client: A Boto3 Amazon Rekognition Custom Labels client.
    :param source_project_arn: The ARN of the source project that contains the
        model that you want to copy.
    :param source_project_version_arn: The ARN of the model version that you want
        to copy.
    :param destination_project_arn: The ARN of the project that you want to copy
        the model
        to.
    :param training_results: The Amazon S3 location where training results for the
        model
        should be stored.
    :return: The model status and version.
    """
    try:
        logger.info("Copying model...%s from %s to %s ",
        source_project_version_arn,
                    source_project_arn,
                    destination_project_arn)

        output_bucket, output_folder = training_results.replace(
            "s3://", "").split("/", 1)
        output_config = {"S3Bucket": output_bucket,
                         "S3KeyPrefix": output_folder}

        response = rekognition_client.copy_project_version(
            DestinationProjectArn=destination_project_arn,
            OutputConfig=output_config,
            SourceProjectArn=source_project_arn,
            SourceProjectVersionArn=source_project_version_arn,
            VersionName=destination_version_name
        )

        destination_model_arn = response["ProjectVersionArn"]

        logger.info("Destination model ARN: %s", destination_model_arn)

        # Wait until training completes.
    
```

```
        finished = False
        status = "UNKNOWN"
        while finished is False:
            model_description =
rekognition_client.describe_project_versions(ProjectArn=destination_project_arn,
VersionNames=[destination_version_name])
            status = model_description["ProjectVersionDescriptions"][0]["Status"]

            if status == "COPYING_IN_PROGRESS":
                logger.info("Model copying in progress...")
                time.sleep(60)
                continue

            if status == "COPYING_COMPLETED":
                logger.info("Model was successfully copied.")

            if status == "COPYING_FAILED":
                logger.info(
                    "Model copy failed: %s ",
                    model_description["ProjectVersionDescriptions"][0]
["StatusMessage"])

                finished = True
        except ClientError:
            logger.exception("Couldn't copy model.")
            raise
    else:
        return destination_model_arn, status

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "source_project_arn", help="The ARN of the project that contains the model
that you want to copy."
    )

    parser.add_argument(
        "source_project_version_arn", help="The ARN of the model version that you
want to copy."
    )

    parser.add_argument(
        "destination_project_arn", help="The ARN of the project which receives the
copied model."
    )

    parser.add_argument(
        "destination_version_name", help="The version name for the model in the
destination project."
    )

    parser.add_argument(
        "training_results", help="The S3 location in the destination account that
receives the training results for the copied model."
    )

def main():
```

```
logging.basicConfig(level=logging.INFO,
                    format="%(levelname)s: %(message)s")

try:

    # get command line arguments
    parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
    add_arguments(parser)
    args = parser.parse_args()

    print(
        f"Copying model version {args.source_project_version_arn} to project
{args.destination_project_arn}")

    rek_client = boto3.client('rekognition')

    # Copy the model.

    model_arn, status = copy_model(rek_client,
                                    args.source_project_arn,
                                    args.source_project_version_arn,
                                    args.destination_project_arn,
                                    args.training_results,
                                    args.destination_version_name,
                                    )

    print(f"Finished copying model: {model_arn}")
    print(f"Status: {status}")

except ClientError as err:
    logger.exception(f"Problem copying model: {err}")
    print(f"Problem copying model: {err}")
except Exception as err:
    logger.exception(f"Problem training model: {err}")
    print(f"Problem copying model: {err}")

if __name__ == "__main__":
    main()
```

Java 2

Use the following code. Supply the following command line parameters:

- **source_project_arn** — the ARN of the source project in the source AWS account that contains the model version that you want to copy.
- **source_project_version_arn** — the ARN of the model version in the source AWS account that you want to copy.
- **destination_project_arn** — the ARN of the destination project that you want to copy the model to.
- **destination_version_name** — a version name for the model in the destination project.
- **output_bucket** — the S3 bucket that you want the training results for the source model version copied to.
- **output_folder** — the folder in the S3 that you want the training results for the source model version copied to.

//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.

```
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/  
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)  
  
import software.amazon.awssdk.regions.Region;  
import software.amazon.awssdk.services.rekognition.RekognitionClient;  
import software.amazon.awssdk.services.rekognition.model.CopyProjectVersionRequest;  
import  
    software.amazon.awssdk.services.rekognition.model.CopyProjectVersionResponse;  
import  
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsRequest;  
import  
    software.amazon.awssdk.services.rekognition.model.DescribeProjectVersionsResponse;  
import software.amazon.awssdk.services.rekognition.model.OutputConfig;  
import software.amazon.awssdk.services.rekognition.model.ProjectVersionDescription;  
  
import software.amazon.awssdk.services.rekognition.model.RekognitionException;  
  
import java.net.URI;  
import java.net.URISyntaxException;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
public class CopyModel {  
  
    public static final Logger logger =  
Logger.getLogger(CopyModel.class.getName());  
  
    public static ProjectVersionDescription copyMyModel(RekognitionClient  
rekClient,  
        String sourceProjectArn,  
        String sourceProjectVersionArn,  
        String destinationProjectArn,  
        String versionName,  
        String outputBucket,  
        String outputFolder) throws InterruptedException {  
  
        try {  
  
            OutputConfig outputConfig =  
OutputConfig.builder().s3Bucket(outputBucket).s3KeyPrefix(outputFolder).build();  
  
            String[] logArguments = new String[] { versionName, sourceProjectArn,  
destinationProjectArn };  
  
            logger.log(Level.INFO, "Copying model {0} for from project {1} to  
project {2}", logArguments);  
  
            CopyProjectVersionRequest copyProjectVersionRequest =  
CopyProjectVersionRequest.builder()  
                .sourceProjectArn(sourceProjectArn)  
                .sourceProjectVersionArn(sourceProjectVersionArn)  
                .versionName(versionName)  
                .destinationProjectArn(destinationProjectArn)  
                .outputConfig(outputConfig)  
                .build();  
  
            CopyProjectVersionResponse response =  
rekClient.copyProjectVersion(copyProjectVersionRequest);  
  
            logger.log(Level.INFO, "Destination model ARN: {0}",  
response.projectVersionArn());  
            logger.log(Level.INFO, "Copying model...");  
  
            // wait until copying completes.  
  
            boolean finished = false;
```

```
ProjectVersionDescription copiedModel = null;

while (Boolean.FALSE.equals(finished)) {
    DescribeProjectVersionsRequest describeProjectVersionsRequest =
DescribeProjectVersionsRequest.builder()
    .versionNames(versionName)
    .projectArn(destinationProjectArn)
    .build();

    DescribeProjectVersionsResponse describeProjectVersionsResponse =
rekClient
    .describeProjectVersions(describeProjectVersionsRequest);

    for (ProjectVersionDescription projectVersionDescription :
describeProjectVersionsResponse
        .projectVersionDescriptions()) {

        copiedModel = projectVersionDescription;

        switch (projectVersionDescription.status()) {

            case COPYING_IN_PROGRESS:
                logger.log(Level.INFO, "Copying model...");
                Thread.sleep(5000);
                continue;

            case COPYING_COMPLETED:
                finished = true;
                logger.log(Level.INFO, "Copying completed");
                break;

            case COPYING_FAILED:
                finished = true;
                logger.log(Level.INFO, "Copying failed...");
                break;

            default:
                finished = true;
                logger.log(Level.INFO, "Unexpected copy status %s",
                    projectVersionDescription.statusAsString());
                break;
        }
    }
}

logger.log(Level.INFO, "Finished copying model {0} for from project {1}
to project {2}", logArguments);

return copiedModel;
} catch (RekognitionException e) {
    logger.log(Level.SEVERE, "Could not train model: {0}", e.getMessage());
    throw e;
}
}

public static void main(String args[]) {
    String sourceProjectArn = null;
    String sourceProjectVersionArn = null;
    String destinationProjectArn = null;
```

```
String versionName = null;
String bucket = null;
String location = null;

final String USAGE = "\n" + "Usage: "
    + "<source_project_arn> <source_project_version_arn>\n"
<destination_project_arn> <version_name> <output_bucket> <output_folder>\n\n"
    + "Where:\n"
    + "  source_project_arn - The ARN of the project that contains the
model that you want to copy. \n\n"
    + "  source_project_version_arn - The ARN of the project that
contains the model that you want to copy. \n\n"
    + "  destination_project_arn - The ARN of the destination project
that you want to copy the model to. \n\n"
    + "  version_name - A version name for the copied model.\n\n"
    + "  output_bucket - The S3 bucket in which to place the training
output. \n\n"
    + "  output_folder - The folder within the bucket that the
training output is stored in. \n\n";

if (args.length != 6) {
    System.out.println(USAGE);
    System.exit(1);
}

sourceProjectArn = args[0];
sourceProjectVersionArn = args[1];
destinationProjectArn = args[2];
versionName = args[3];
bucket = args[4];
location = args[5];

try {

    // Get the Rekognition client.
    RekognitionClient rekClient = RekognitionClient.builder().build();

    // Copy the model.
    ProjectVersionDescription copiedModel = copyMyModel(rekClient,
        sourceProjectArn,
        sourceProjectVersionArn,
        destinationProjectArn,
        versionName,
        bucket,
        location);

    System.out.println(String.format("Model copied: %s Status: %s",
        copiedModel.projectVersionArn(),
        copiedModel.statusMessage()));

    rekClient.close();

} catch (RekognitionException rekError) {
    logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
    System.exit(1);
} catch (InterruptedException intError) {
    logger.log(Level.SEVERE, "Exception while sleeping: {0}",
intError.getMessage());
    System.exit(1);
}
}
```

Listing project policies (SDK)

You can use the [ListProjectPolicies](#) operation to list the project policies that are attached to an Amazon Rekognition Custom Labels project.

To list the project policies attached to a project (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with `AmazonRekognitionCustomLabelsFullAccess` permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following code to list the project policies.

AWS CLI

Change `project-arn` to the Amazon Resource Name of the project for which you want to list the attached project policies.

```
aws rekognition list-project-policies \
  --project-arn project-arn
```

Python

Use the following code. Supply the following command line parameters:

- `project_arn` — the Amazon Resource Name of the project for which you want to list the attached project policies.

For example: `python list_project_policies.py project_arn`

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

"""
Purpose
Amazon Rekognition Custom Labels model example used in the service documentation:
https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/md-copy-model-
sdk.html
Shows how to list the project policies in an Amazon Rekognition Custom Labels
project.
"""

import boto3
import argparse
import logging
import json
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def display_project_policy(project_policy):
    """
    Displays information about a Custom Labels project policy.
    :param project_policy: The project policy (ProjectPolicy) that you want to
    display information about.
    
```

```
"""
print(f"Policy name: {project_policy['PolicyName']}")
print(f"Project Arn: {project_policy['ProjectArn']}")
print(f"Document: {project_policy['PolicyDocument']}")
print(f"Revision ID: {project_policy['PolicyRevisionId']}")
print()

def list_project_policies(rek_client, project_arn):
    """
    Describes an Amazon Rekognition Custom Labels project, or all projects.
    :param rek_client: The Amazon Rekognition Custom Labels Boto3 client.
    :param project_arn: The Amazon Resource Name of the project you want to use.
    """
    try:
        max_results = 5
        pagination_token = ''
        finished = False

        logger.info(f"Listing project policies in: {project_arn}.")
        print('Projects\n-----')
        while not finished:
            response = rek_client.list_project_policies(
                ProjectArn=project_arn, MaxResults=max_results,
                NextToken=pagination_token)

            for project in response['ProjectPolicies']:
                display_project_policy(project)

            if 'NextToken' in response:
                pagination_token = response['NextToken']
            else:
                finished = True

        logger.info(f"Finished listing project policies.")

    except ClientError as err:
        logger.exception(
            f"Couldn't list policies for - {project_arn}: {err.response['Error']['Message']}")

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """
    parser.add_argument(
        "project_arn", help="The Amazon Resource Name of the project for which you
        want to list project policies."
    )

def main():
    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:
```

```
# get command line arguments
parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
add_arguments(parser)
args = parser.parse_args()

print(f"Listing project policies in: {args.project_arn}")

# List the project policies.

rek_client = boto3.client('rekognition')

list_project_policies(rek_client,
                      args.project_arn)

except ClientError as err:
    logger.exception(f"Problem list project_policies: {err}")
    print(f"Problem list project_policies: {err}")

if __name__ == "__main__":
    main()
```

Java 2

Use the following code. Supply the following command line parameters:

- project_arn — The ARN of the project that has the project policies you want to list.

```
//Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import java.net.URI;
import java.net.URISyntaxException;
import java.util.logging.Level;
import java.util.logging.Logger;

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
software.amazon.awssdk.services.rekognition.model.ListProjectPoliciesRequest;
import
software.amazon.awssdk.services.rekognition.model.ListProjectPoliciesResponse;
import software.amazon.awssdk.services.rekognition.model.ProjectPolicy;
import software.amazon.awssdk.services.rekognition.model.RekognitionException;

public class ListProjectPolicies {

    public static final Logger logger =
Logger.getLogger(ListProjectPolicies.class.getName());

    public static void listMyProjectPolicies(RekognitionClient rekClient, String
projectArn) {

        try {

            logger.log(Level.INFO, "Listing project policies for project: {0}",
projectArn);

            // List the project policies.

            Boolean finished = false;
            String nextToken = null;
```

```
        while (Boolean.FALSE.equals(finished)) {

            ListProjectPoliciesRequest listProjectPoliciesRequest =
ListProjectPoliciesRequest.builder()
                .maxResults(5)
                .projectArn(projectArn)
                .nextToken(nextToken)
                .build();

            ListProjectPoliciesResponse response =
rekClient.listProjectPolicies(listProjectPoliciesRequest);

            for (ProjectPolicy projectPolicy : response.projectPolicies()) {

                System.out.println(String.format("Name: %s",
projectPolicy.policyName()));
                System.out.println(String.format("Revision ID: %s\n",
projectPolicy.policyRevisionId()));

            }

            nextToken = response.nextToken();

            if (nextToken == null) {
                finished = true;
            }

        }

        logger.log(Level.INFO, "Finished listing project policies for project:
{0}", projectArn);

    } catch (
        RekognitionException e) {
        logger.log(Level.SEVERE, "Client error occurred: {0}", e.getMessage());
        throw e;
    }

}

public static void main(String args[]) {

    final String USAGE = "\n" + "Usage: " + "<project_arn> \n\n" + "Where:\n"
        + "    project_arn - The ARN of the project with the project
policies that you want to list.\n\n";
    ;

    if (args.length != 1) {
        System.out.println(USAGE);
        System.exit(1);
    }

    String projectArn = args[0];

    try {

        RekognitionClient rekClient = RekognitionClient.builder().build();

        // List the project policies.
        listMyProjectPolicies(rekClient, projectArn);

        rekClient.close();

    }
```

```
        } catch (RekognitionException rekError) {
            logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
            System.exit(1);
        }
    }
}
```

Deleting a project policy (SDK)

You can use the [DeleteProjectPolicy](#) operation to delete a revision of an existing project policy from an Amazon Rekognition Custom Labels project. If you want to delete all revisions of a project policy that are attached to a project, use [ListProjectPolicies](#) to get the revision IDs of each project policy attached to the project. Then call [DeletePolicy](#) for each policy name.

To delete a revision of a project policy (SDK)

1. If you haven't already:
 - a. Create or update an IAM user with `AmazonRekognitionCustomLabelsFullAccess` permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following code to delete a project policy.

`DeletePolicy` takes `ProjectARN`, `PolicyName` and `PolicyRevisionId`. `ProjectARN` and `PolicyName` are required for this API. `PolicyRevisionId` is optional, but can be included for the purposes of atomic updates.

AWS CLI

Change the following values:

- `policy-name` to the name of the project policy that you want to delete.
- `policy-revision-id` to the revision ID of the project policy that you want to delete.
- `project-arn` to the Amazon Resource Name of the project that contains the revision of the project policy that you want to delete.

```
aws rekognition delete-project-policy \
--policy-name policy-name \
--policy-revision-id policy-revision-id \
--project-arn project-arn
```

Python

Use the following code. Supply the following command line parameters:

- `policy-name` – The name of the project policy that you want to delete.
- `project-arn` – The Amazon Resource Name of the project that contains the project policy that you want to delete.
- `policy-revision-id` – The revision ID of the project policy that you want to delete.

For example: `python delete_project_policy.py policy_name project_arn policy_revision_id`

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

"""
Purpose
Amazon Rekognition Custom Labels model example used in the service documentation:
https://docs.aws.amazon.com/rekognition/latest/customlabels-dg/md-copy-model-
sdk.html
Shows how to delete a revision of a project policy.
"""

import boto3
import argparse
import logging
import time
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)

def delete_project_policy(rekognition_client, policy_name, project_arn,
    policy_revision_id=None):
    """
    Deletes a project policy.

    :param rekognition_client: A Boto3 Amazon Rekognition client.
    :param policy_name: The name of the project policy that you want to delete.
    :param policy_revision_id: The revision ID for the project policy that you want
    to delete.
    :param project_arn: The Amazon Resource Name of the project that contains the
    project policy
    that you want to delete.
    """
    try:
        logger.info("Deleting project policy: %s", policy_name)

        if policy_revision_id is None:
            rekognition_client.delete_project_policy(
                PolicyName=policy_name,
                ProjectArn=project_arn)

        else:
            rekognition_client.delete_project_policy(
                PolicyName=policy_name,
                PolicyRevisionId=policy_revision_id,
                ProjectArn=project_arn)

        logger.info("Deleted project policy: %s", policy_name)
    except ClientError:
        logger.exception("Couldn't delete project policy.")
        raise

def confirm_project_policy_deletion(policy_name):
    """
    Confirms deletion of the project policy. Returns True if delete entered.

    :param model_arn: The ARN of the model that you want to delete.
    """
    print(
```

```
f"Are you sure you want to delete project policy {policy_name} ?\n",
policy_name)

delete = input("Enter delete to delete your project policy: ")
if delete == "delete":
    return True
else:
    return False

def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "policy_name", help="The ARN of the project that contains the project
policy that you want to delete."
    )

    parser.add_argument(
        "project_arn", help="The ARN of the project project policy you want to
delete."
    )

    parser.add_argument(
        "--policy_revision_id", help="(Optional) The revision ID of the project
policy that you want to delete.",
        required=False
    )

def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        if confirm_project_policy_deletion(args.policy_name) == True:
            print(f"Deleting project policy: {args.policy_name}")

            rek_client = boto3.client('rekognition')

            # Delete the project policy.

            delete_project_policy(rek_client,
                                  args.policy_name,
                                  args.project_arn,
                                  args.policy_revision_id)

            print(f"Finished deleting project policy: {args.policy_name}")
        else:
            print(f"Not deleting project policy {args.policy_name}")
    except ClientError:
        logger.exception(
            "Couldn't delete project policy in %s.", args.policy_name)
        raise
```

```
if __name__ == "__main__":
    main()
```

Java 2

Use the following code. Supply the following command line parameters:

- **policy-name** – The name of the project policy that you want to delete.
- **project-arn** – The Amazon Resource Name of the project that contains the project policy that you want to delete.
- **policy-revision-id** – The revision ID of the project policy that you want to delete.

```
//Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
//PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/
amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)

import java.util.logging.Level;
import java.util.logging.Logger;

import software.amazon.awssdk.services.rekognition.RekognitionClient;
import
    software.amazon.awssdk.services.rekognition.model.DeleteProjectPolicyRequest;

import software.amazon.awssdk.services.rekognition.model.RekognitionException;

public class DeleteProjectPolicy {

    public static final Logger logger =
Logger.getLogger(DeleteProjectPolicy.class.getName());

    public static void deleteMyProjectPolicy(RekognitionClient rekClient, String
projectArn,
        String projectPolicyName,
        String projectPolicyRevisionId)
        throws InterruptedException {

        try {
            String[] logArguments = new String[] { projectPolicyName,
projectPolicyRevisionId };

            logger.log(Level.INFO, "Deleting: Project policy: {0} revision: {1}",
logArguments);

            // Delete the project policy.

            DeleteProjectPolicyRequest deleteProjectPolicyRequest =
DeleteProjectPolicyRequest.builder()
                .policyName(projectPolicyName)
                .policyRevisionId(projectPolicyRevisionId)
                .projectArn(projectArn).build();

            rekClient.deleteProjectPolicy(deleteProjectPolicyRequest);

            logger.log(Level.INFO, "Deleted: Project policy: {0} revision: {1}",
logArguments);

        } catch (
            RekognitionException e) {
            logger.log(Level.SEVERE, "Client error occurred: {0}", e.getMessage());
            throw e;
        }
    }
}
```

```
}

public static void main(String args[]) {

    final String USAGE = "\n" + "Usage: " + "<project_arn>\n<project_policy_name> <project_policy_revision_id>\n\n"
        + "Where:\n"
        + "  project_arn - The ARN of the project that has the project
policy that you want to delete.\n\n"
        + "  project_policy_name - The name of the project policy that you
want to delete.\n\n"
        + "  project_policy_revision_id - The revision of the project
policy that you want to delete.\n\n";

    if (args.length != 3) {
        System.out.println(USAGE);
        System.exit(1);
    }

    String projectArn = args[0];
    String projectPolicyName = args[1];
    String projectPolicyRevisionId = args[2];

    try {

        RekognitionClient rekClient = RekognitionClient.builder().build();

        // Delete the project policy.
        deleteMyProjectPolicy(rekClient, projectArn, projectPolicyName,
        projectPolicyRevisionId);

        System.out.println(String.format("project policy deleted: %s revision:
%s", projectPolicyName,
        projectPolicyRevisionId));

        rekClient.close();

    } catch (RekognitionException rekError) {
        logger.log(Level.SEVERE, "Rekognition client error: {0}",
rekError.getMessage());
        System.exit(1);
    }

    catch (InterruptedException intError) {
        logger.log(Level.SEVERE, "Exception while sleeping: {0}",
intError.getMessage());
        System.exit(1);
    }

}
}
```

Examples

This section contains information about examples that you can use with Amazon Rekognition Custom Labels.

Example	Description
Model feedback solution (p. 309)	Shows how to improve a model using human verification to create a new training dataset.
Amazon Rekognition Custom Labels demonstration (p. 309)	Demonstration of a user interface that displays the results of a call to <code>DetectCustomLabels</code> .
Video analysis (p. 310)	Shows how you can use <code>DetectCustomLabels</code> with frames extracted from a video.
Creating an AWS Lambda function (p. 311)	Shows how you can use <code>DetectCustomLabels</code> with a Lambda function.
Creating a manifest file from a CSV file (p. 313)	Shows how to use a CSV file to create a manifest file suitable for finding objects, scenes, and concepts (p. 27) associated with an entire image (classification).

Model feedback solution

The Model Feedback solution enables you to give feedback on your model's predictions and make improvements by using human verification. Depending on the use case, you can be successful with a training dataset that has only a few images. A larger annotated training set might be required to build a more accurate model. The Model Feedback solution allows you to create a larger dataset through model assistance.

To install and configure the Model Feedback solution, see [Model Feedback Solution](#).

The workflow for continuous model improvement is as follows:

1. Train the first version of your model (possibly with a small training dataset).
2. Provide an unannotated dataset for the Model Feedback solution.
3. The Model Feedback solution uses the current model. It starts human verification jobs to annotate a new dataset.
4. Based on human feedback, the Model Feedback solution generates a manifest file that you use to create a new model.

Amazon Rekognition Custom Labels demonstration

The Amazon Rekognition Custom Labels Demonstration shows a user interface that analyzes images from your local computer by using the `DetectCustomLabels` API.

The application shows you information about the Amazon Rekognition Custom Labels models in your account. After you select a running model, you can analyze an image from your local computer. If necessary, the application allows you to start a model. You can also stop a running model. The application shows integration with other AWS Services such as Amazon Cognito, Amazon S3, and Amazon CloudFront.

For more information, see [Amazon Rekognition Custom Labels Demo](#).

Video analysis

The following example shows how you can use `DetectCustomLabels` with frames extracted from a video. The code has been tested with video files in `mov` and `mp4` format.

Using `DetectCustomLabels` with captured frames

1. If you haven't already:
 - a. Create or update an IAM user with `AmazonRekognitionFullAccess` and `AmazonS3ReadOnlyAccess` permissions. For more information, see [Step 2: Create an IAM administrator user and group \(p. 4\)](#).
 - b. Install and configure the AWS CLI and the AWS SDKs. For more information, see [Step 3: Set Up the AWS CLI and AWS SDKs \(p. 5\)](#).
2. Use the following example code. Change the value of `videoFile` to the name of a video file. Change the value of `projectVersionArn` to the Amazon Resource Name (ARN) of your Amazon Rekognition Custom Labels model.

```
#Copyright 2020 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)  
  
import json
import boto3
import cv2
import math
import io  
  
def analyzeVideo():
    videoFile = "video file"
    projectVersionArn = "project arn"

   rekognition = boto3.client('rekognition')
    customLabels = []
    cap = cv2.VideoCapture(videoFile)
    frameRate = cap.get(5) #frame rate
    while(cap.isOpened()):
        frameId = cap.get(1) #current frame number
        print("Processing frame id: {}".format(frameId))
        ret, frame = cap.read()
        if (ret != True):
            break
        if (frameId % math.floor(frameRate) == 0):
            hasFrame, imageBytes = cv2.imencode(".jpg", frame)

            if(hasFrame):
                response = rekognition.detect_custom_labels(
                    Image={
                        'Bytes': imageBytes.tobytes(),
                    },
                    ProjectVersionArn = projectVersionArn
```

```
)  
  
for elabel in response["CustomLabels"]:  
    elabel["Timestamp"] = (frameId/frameRate)*1000  
    customLabels.append(elabel)  
  
print(customLabels)  
  
with open(videoFile + ".json", "w") as f:  
    f.write(json.dumps(customLabels))  
  
cap.release()  
  
analyzeVideo()
```

Creating an AWS Lambda function

You can call Amazon Rekognition Custom Labels API operations from within an AWS Lambda function. The following instructions show how to create a Lambda function in Python that calls `DetectCustomLabels`. It returns a list of custom label objects. To run this example, you need an Amazon S3 bucket that contains an image in PNG or JPEG format.

Step 1: Create an AWS Lambda deployment package

1. Open a command window.
2. Enter the following commands to create a deployment package with the most recent version of the AWS SDK.

```
pip install boto3 --target python/.  
zip boto3-layer.zip -r python/
```

Step 2: Create an AWS Lambda function (console)

1. Sign in to the AWS Management Console and open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. Choose **Create function**. For more information, see [Create a Lambda Function with the Console](#).
3. Choose the following options.
 - Choose **Author from scratch**.
 - Enter a value for **Function name**.
 - For **Runtime** choose **Python 3.7** or **Python 3.6**.
 - For **Choose or create an execution role**, choose **Create a new role with basic Lambda permissions**.
4. Choose **Create function** to create the AWS Lambda function.
5. Open the IAM console at <https://console.aws.amazon.com/iam/>.
6. From the navigation pane, choose **Roles**.
7. From the resources list, choose the IAM role that AWS Lambda created for you. The role name is prepended with the name of your Lambda function.
8. In the **Permissions** tab, choose **Attach policies**.
9. Add the *AmazonRekognitionFullAccess* and *AmazonS3ReadOnlyAccess* policies.
10. Choose **Attach Policy**.

Step 3: Create and add a layer (console)

1. Open the AWS Lambda console at <https://console.aws.amazon.com/lambda/>.
2. In the navigation pane, choose **Layers**.
3. Choose **Create layer**.
4. Enter values for **Name** and **Description**.
5. For **Code entry type**, choose **Upload .zip file** and choose **Upload**.
6. In the dialog box, choose the zip file (boto3-layer.zip) that you created in [Step 1: Create an AWS Lambda deployment package \(p. 311\)](#).
7. For compatible runtimes, choose the runtime that you chose in [Step 2: Create an AWS Lambda function \(console\) \(p. 311\)](#).
8. Choose **Create** to create the layer.
9. Choose the navigation pane menu icon.
10. In the navigation pane, choose **Functions**.
11. In the resources list, choose the function that you created in [Step 2: Create an AWS Lambda function \(console\) \(p. 311\)](#).
12. In the **Designer** section of the **Configuration** tab, choose **Layers** (under your Lambda function name).
13. In the **Layers** section, choose **Add a layer**.
14. Choose **Select from list of runtime compatible layers**.
15. In **Compatible layers**, choose the **Name** and **Version** of the layer name and version that you created in step 3.
16. Choose **Add**.

Step 4: Add Python code (console)

1. In **Designer**, choose your function name.
2. In the function code editor, add the following to the file **lambda_function.py**. Change the values of bucket and photo to your bucket and image. Change the value of model_arn to the Amazon Resource Name (ARN) of your Amazon Rekognition Custom Labels model.

```
import json
import boto3

def lambda_handler(event, context):

    bucket="bucket"
    photo="image"
    model_arn='model_arn'

    client=boto3.client('rekognition')

    #process using S3 object

    response = client.detect_custom_labels(Image={'S3Object': {'Bucket': bucket,
    'Name': photo}},
    MinConfidence=30,
    ProjectVersionArn=model_arn)

    #Get the custom labels
    labels=response['CustomLabels']

    return {
        'statusCode': 200,
        'body': json.dumps(labels)
```

}

3. Choose **Save** to save your Lambda function.

Step 5: Test your Lambda function (console)

1. Choose **Test**.
2. Enter a value for **Event name**.
3. Choose **Create**.
4. Choose **Test**. The Lambda function is invoked. The output is displayed in the **Execution results** pane of the code editor. The output is a list of custom labels.

If the AWS Lambda function returns a timeout error, a call to an Amazon Rekognition Custom Labels API operation might be the cause. For information about extending the timeout period for an AWS Lambda function, see [AWS Lambda Function Configuration](#).

For information about invoking a Lambda function from your code, see [Invoking AWS Lambda Functions](#).

Creating a manifest file from a CSV file

This example Python script simplifies the creation of a manifest file by using a Comma Separated Values (CSV) file to label images. You create the CSV file. The manifest file is suitable for [Multi-label image classification \(p. 12\)](#) or [Multi-label image classification \(p. 12\)](#). For more information, see [Find objects, scenes, and concepts \(p. 27\)](#).

Note

This script doesn't create a manifest file suitable for finding [object locations \(p. 28\)](#) or for finding [brand locations \(p. 29\)](#).

A manifest file describes the images used to train a model. For example, image locations and labels assigned to images. A manifest file is made up of one or more JSON lines. Each JSON line describes a single image. For more information, see [the section called "Image-Level labels in manifest files" \(p. 249\)](#).

A CSV file represents tabular data over multiple rows in a text file. Fields on a row are separated by commas. For more information, see [comma separated values](#). For this script, each row in your CSV file represents a single image and maps to a JSON Line in the manifest file. To create a CSV file for a manifest file that supports [Multi-label image classification \(p. 12\)](#), you add one or more image-level labels to each row. To create a manifest file suitable for [Image classification \(p. 12\)](#), you add a single image-level label to each row.

For example, The following CSV file describes the images in the [Multi-label image classification \(p. 12\)](#) (*Flowers*) *Getting started* project.

```
camellia1.jpg,camellia,with_leaves
camellia2.jpg,camellia,with_leaves
camellia3.jpg,camellia,without_leaves
helleborus1.jpg,helleborus,without_leaves,not_fully_grown
helleborus2.jpg,helleborus,with_leaves,fully_grown
helleborus3.jpg,helleborus,with_leaves,fully_grown
jonquil1.jpg,jonquil,with_leaves
jonquil2.jpg,jonquil,with_leaves
jonquil3.jpg,jonquil,with_leaves
jonquil4.jpg,jonquil,without_leaves
mauve_honey_myrtle1.jpg,mauve_honey_myrtle,without_leaves
mauve_honey_myrtle2.jpg,mauve_honey_myrtle,with_leaves
mauve_honey_myrtle3.jpg,mauve_honey_myrtle,with_leaves
mediterranean_spurge1.jpg,mediterranean_spurge,with_leaves
```

```
mediterranean_spurge2.jpg,mediterranean_spurge,without_leaves
```

The script generates JSON Lines for each row. For example, the following is the JSON Line for the first row (camellia1.jpg, camellia, with_leaves).

```
{"source-ref": "s3://bucket/flowers/train/camellia1.jpg", "camellia": 1, "camellia-metadata": {"confidence": 1, "job-name": "labeling-job/camellia", "class-name": "camellia", "human-annotated": "yes", "creation-date": "2022-01-21T14:21:05", "type": "groundtruth/image-classification"}, "with_leaves": 1, "with_leaves-metadata": {"confidence": 1, "job-name": "labeling-job/with_leaves", "class-name": "with_leaves", "human-annotated": "yes", "creation-date": "2022-01-21T14:21:05", "type": "groundtruth/image-classification"}}
```

In the example CSV, the Amazon S3 path to the image is not present. If your CSV file doesn't include the Amazon S3 path for the images, use the `--s3_path` command line argument to specify the Amazon S3 path to the image.

The script records the first entry for each image in a deduplicated image CSV file. The deduplicated image CSV file contains a single instance of each image found in the input CSV file. Further occurrences of an image in the input CSV file are recorded in a duplicate image CSV file. If the script finds duplicate images, review the duplicate image CSV file and update the deduplicated image CSV file as necessary. Rerun the script with the deduplicated file. If no duplicates are found in the input CSV file, the script deletes the deduplicated image CSV file and duplicate image CSV file, as they are empty.

In this procedure, you create the CSV file and run the Python script to create the manifest file.

To create a manifest file from a CSV file

1. Create a CSV file with the following fields in each row (one row per image). Don't add a header row to the CSV file.

Field 1	Field 2	Field n
The image name or the Amazon S3 path the image. For example, <code>s3://my-bucket/flowers/train/camellia1.jpg</code> . You can't have a mixture of images with the Amazon S3 path and images without.	The first image level label for the image.	One or more additional image-level labels separated by commas. Add only if you want to create a manifest file that supports Multi-label image classification (p. 12) .

For example `camellia1.jpg, camellia, with_leaves` or `s3://my-bucket/flowers/train/camellia1.jpg, camellia, with_leaves`

2. Save the CSV file.
3. Run the following Python script. Supply the following arguments:
 - `csv_file` – The CSV file that you created in step 1.
 - `manifest_file` – The name of the manifest file that you want to create.
 - (Optional) `--s3_path s3://path_to_folder/` – The Amazon S3 path to add to the image file names (field 1). Use `--s3_path` if the images in field 1 don't already contain an S3 path.

```
# Copyright 2021 Amazon.com, Inc. or its affiliates. All Rights Reserved.  
# PDX-License-Identifier: MIT-0 (For details, see https://github.com/awsdocs/amazon-rekognition-custom-labels-developer-guide/blob/master/LICENSE-SAMPLECODE.)
```

```
from datetime import datetime, timezone
import argparse
import logging
import csv
import os

"""
Purpose
Amazon Rekognition Custom Labels model example used in the service documentation:
Shows how to create an image-level (classification) manifest file from a CSV file.
You can specify multiple image level labels per image.
CSV file format is
image,label,label,..
If necessary, use the bucket argument to specify the S3 bucket folder for the images.

"""

logger = logging.getLogger(__name__)

def check_duplicates (csv_file, deduplicated_file, duplicates_file):
    """
    Checks for duplicate images in a CSV file. If duplicate images
    are found, deduplicated_file is the deduplicated CSV file - only the first
    occurrence of a duplicate is recorded. Other duplicates are recorded in
    duplicates_file.
    :param csv_file: The source CSV file
    :param deduplicated_file: The deduplicated CSV file to create. If no duplicates are
    found
        this file is removed.
    :param duplicates_file: The duplicate images CSV file to create. If no duplicates
    are found
        this file is removed.
    :return: True if duplicates are found, otherwise false.
    """
    logger.info(f"Deduplicating {csv_file}.")  

    duplicates_found = False  

    #Find duplicates
    with open(csv_file, 'r') as f,\n        open(deduplicated_file, 'w') as dedup,\n        open(duplicates_file, 'w') as duplicates:  

        reader = csv.reader(f, delimiter=',')
        dedup_writer = csv.writer(dedup)
        duplicates_writer = csv.writer(duplicates)

        entries = set()
        for row in reader:
            #Skip empty lines
            if row==[]:
                continue

            key = row[0]
            if key not in entries:
                dedup_writer.writerow(row)
                entries.add(key)
            else:
                duplicates_writer.writerow(row)
                duplicates_found = True

    if duplicates_found:
        logger.info(f"Duplicates found check {duplicates_file}.")
```

```
else:
    os.remove(duplicates_file)
    os.remove(deduplicated_file)

return duplicates_found

def create_manifest_file(csv_file, manifest_file, s3_path):
    """
    Reads a CSV file and creates a Custom Labels classification manifest file
    :param csv_file: The source CSV file
    :param manifest_file: The name of the manifest file to create.
    :param s3_path: The S3 path to the folder that contains the images.
    """
    logger.info(f"Processing CSV file {csv_file}.")  
  
    image_count = 0
    label_count = 0  
  
    with open(csv_file, newline='') as csvfile, open(manifest_file, "w") as output_file:  
  
        image_classifications = csv.reader(
            csvfile, delimiter=',', quotechar='|')  
  
        # process each row (image) in CSV file
        for row in image_classifications:
            source_ref = str(s3_path)+row[0]
            image_json = '{"source-ref": "' + source_ref + '",'
            label_json = ''
            last_line = ''
            image_count += 1  
  
            # process each image level label
            for index in range(1, len(row)):
                image_level_label = row[index]  
  
                #Skip empty columns
                if image_level_label=='':
                    continue
                label_count += 1  
  
                label_json += '' + image_level_label + ': 1, \'\
                    '' + image_level_label + '-metadata': '\
                    '{"confidence": 1, \'\
                    "'job-name": "labeling-job/' + image_level_label + '", '\
                    "'class-name": "' + image_level_label + '", '\
                    "'human-annotated": "yes", '\
                    "'creation-date": "' + datetime.now(timezone.utc).strftime('%Y-%m-%dT%H:%M:%S.%f') + '", '  
  
                last_line = {"type": "groundtruth/image-classification"},'
            if index == len(row)-1:
                last_line = {"type": "groundtruth/image-classification"}'  
  
            label_json += last_line
            json_line = image_json + label_json + '\n'
            # write the image JSON Line
            output_file.write(json_line)  
  
        output_file.close()
        logger.info(f"Finished creating manifest file {manifest_file}.\n"
                   f"Images: {image_count}\nLabels: {label_count}")
    return image_count, label_count
```

```
def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "csv_file", help="The CSV file that you want to process."
    )

    parser.add_argument(
        "--s3_path", help="The S3 bucket and folder path for the images."
        " If not supplied, column 1 is assumed to include the S3 path.", required=False
    )

def main():
    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # get command line arguments
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        s3_path=args.s3_path
        if s3_path is None:
            s3_path=''

        csv_file = args.csv_file
        manifest_file = os.path.splitext(csv_file)[0] + '.manifest'
        duplicates_file = f'duplicates-{csv_file}'
        deduplicated_file = f'deduplicated-{csv_file}'

        #Create mainfest file if there are no duplicate images.
        if check_duplicates(csv_file, deduplicated_file, duplicates_file):
            print(
                f'Duplicates found. Use {duplicates_file} to view duplicates and then
                update {deduplicated_file}. ')
                print(f'{deduplicated_file} contains the first occurence of a duplicate. \
                    'Update as necessary with the correct label information.')
                print(f'Re-run the script with {deduplicated_file}')
        else:
            print('No duplicates found. Creating manifest file')

            image_count, label_count = create_manifest_file(csv_file,
                                                manifest_file,
                                                s3_path)

            print(f"Finished creating manifest file: {manifest_file} \
                \nImages: {image_count}\nLabels: {label_count}")

    except FileNotFoundError as err:
        logger.exception(f"File not found.:{err}")
        print(f"File not found: {err}. Check your input CSV file")

    except Exception as err:
        logger.exception(f"An error occurred:{err}")
        print(f"An error occurred:{err}")

if __name__ == "__main__":
```

```
main()
```

4. If you plan to use a test dataset, repeat steps 1–3 to create a manifest file for your test dataset.
5. If necessary, copy the images to the Amazon S3 bucket path that you specified in column 1 of the CSV file (or specified in the `--s3_path` command line). You can use the following AWS S3 command.

```
aws s3 cp --recursive your-local-folder s3://your-target-S3-location
```

6. Follow the instructions at [Creating a dataset with a manifest file \(p. 243\)](#) to create a dataset.

Security

You can secure the management of your models and the `DetectCustomLabels` API that your customers use to detect custom labels.

For more information about securing Amazon Rekognition, see [Amazon Rekognition Security](#).

Securing Amazon Rekognition Custom Labels projects

You can secure your Amazon Rekognition Custom Labels projects by specifying the resource-level permissions that are specified in identity-based policies. For more information, see [Identity-Based Policies and Resource-Based Policies](#).

The Amazon Rekognition Custom Labels resources that you can secure are:

Resource	Amazon Resource Name Format
Project	arn:aws:rekognition:*:project/ <i>project_name</i> /datetime
Model	arn:aws:rekognition:*:project/ <i>project_name</i> /version/ <i>name</i> /datetime

The following example policy shows how to give an identity permission to:

- Describe all projects.
- Create, start, stop, and use a specific model for inference.
- Create a project. Create and describe a specific model.
- Deny the creation of a specific project.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllResources",  
            "Effect": "Allow",  
            "Action": "rekognition:DescribeProjects",  
            "Resource": "*"  
        },  
        {  
            "Sid": "SpecificProjectVersion",  
            "Effect": "Allow",  
            "Action": [  
                "rekognition:StopProjectVersion",  
                "rekognition:StartProjectVersion",  
                "rekognition:DetectCustomLabels",  
            ]  
        }  
    ]  
}
```

```
        "rekognition:CreateProjectVersion"
    ],
    "Resource": "arn:aws:rekognition:*:project/MyProject/version/MyVersion/*"
},
{
    "Sid": "SpecificProject",
    "Effect": "Allow",
    "Action": [
        "rekognition:CreateProject",
        "rekognition:DescribeProjectVersions",
        "rekognition:CreateProjectVersion"
    ],
    "Resource": "arn:aws:rekognition:*:project/MyProject/*"
},
{
    "Sid": "ExplicitDenyCreateProject",
    "Effect": "Deny",
    "Action": [
        "rekognition:CreateProject"
    ],
    "Resource": ["arn:aws:rekognition:*:project/SampleProject/*"]
}
]
```

Securing DetectCustomLabels

The identity used to detect custom labels might be different from the identity that manages Amazon Rekognition Custom Labels models.

You can secure access an identity's access to DetectCustomLabels by applying a policy to the identity. The following example restricts access to DetectCustomLabels only and to a specific model. The identity doesn't have access to any of the other Amazon Rekognition operations.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rekognition:DetectCustomLabels"
            ],
            "Resource": "arn:aws:rekognition:*:project/MyProject/version/MyVersion/*"
        }
    ]
}
```

AWS managed policy: AmazonRekognitionCustomLabelsFullAccess

To add permissions to users, groups, and roles, it is easier to use AWS 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 AWS managed policies.

These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies](#) in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see [AWS managed policies for job functions](#) in the *IAM User Guide*.

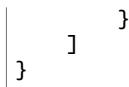
Using AmazonRekognitionCustomLabelsFullAccess

This policy is for Amazon Rekognition Custom Labels users. Use the **AmazonRekognitionCustomLabelsFullAccess** policy to allow users full access to the Amazon Rekognition Custom Labels API and full access to the console buckets created by the Amazon Rekognition Custom Labels console.

Permissions details

This policy includes the following permissions.

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": [  
        "s3>ListBucket",  
        "s3>ListAllMyBuckets",  
        "s3>GetBucketAcl",  
        "s3>GetBucketLocation",  
        "s3>GetObject",  
        "s3>GetObjectAcl",  
        "s3>GetObjectTagging",  
        "s3>GetObjectVersion",  
        "s3>PutObject"  
      ],  
      "Resource": "arn:aws:s3:::*custom-labels*"  
    },  
    {  
      "Effect": "Allow",  
      "Action": [  
        "rekognition>CreateProject",  
        "rekognition>CreateProjectVersion",  
        "rekognition>StartProjectVersion",  
        "rekognition>StopProjectVersion",  
        "rekognition>DescribeProjects",  
        "rekognition>DescribeProjectVersions",  
        "rekognition>DetectCustomLabels",  
        "rekognition>DeleteProject",  
        "rekognition>DeleteProjectVersion",  
        "rekognition>TagResource",  
        "rekognition>UntagResource",  
        "rekognition>ListTagsForResource"  
      ],  
      "Resource": "*"  
    }  
  ]  
}
```



Amazon Rekognition Custom Labels updates to AWS managed policies

View details about updates to AWS managed policies for Amazon Rekognition Custom Labels since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon Rekognition Custom Labels Document history page.

Change	Description	Date
AmazonRekognitionCustomLabels tagging update	Amazon Rekognition Custom Labels added new tagging actions.	April 2, 2021
Amazon Rekognition Custom Labels started tracking changes	Amazon Rekognition Custom Labels started tracking changes for its AWS managed policies.	April 2, 2021

Guidelines and quotas in Amazon Rekognition Custom Labels

The following sections provide guidelines and quotas when using Amazon Rekognition Custom Labels.

Supported Regions

For a list of AWS Regions where Amazon Rekognition Custom Labels is available, see [AWS Regions and Endpoints](#) in the *Amazon Web Services General Reference*.

Quotas

The following is a list of limits in Amazon Rekognition Custom Labels. For information about limits you can change, see [AWS Service Limits](#). To change a limit, see [Create Case](#).

Training

- Supported file formats are PNG and JPEG image formats.
- Maximum number of training datasets in a version of a model is 1.
- Maximum dataset manifest file size is 1 GB.
- Minimum number of unique labels per Objects, Scenes, and Concepts (classification) dataset is 2.
- Minimum number of unique labels per Object Location (detection) dataset is 1.
- Maximum number of unique labels per manifest is 250.
- Minimum number of images per label is 1.
- Maximum number of images per Object Location (detection) dataset is 250,000.

The limit for *Asia Pacific (Mumbai)* and *Europe (London)* AWS Regions is 28,000 images.

- Maximum number of images per Objects, Scenes, and Concepts (classification) dataset is 500,000. The default is 250,000. To request an increase, see [Create Case](#).

The limit for *Asia Pacific (Mumbai)* and *Europe (London)* AWS Regions is 28,000 images. You can't request a limit increase.

- Maximum number of labels per image is 50.
- Minimum number of bounding boxes in an image is 0.
- Maximum number of bounding boxes in an image is 50.
- Minimum image dimension of image file in an Amazon S3 bucket is 64 pixels x 64 pixels.
- Maximum image dimension of image file in an Amazon S3 bucket is 4096 pixels x 4096 pixels.
- Maximum file size for an image in an Amazon S3 bucket is 15 MB.
- Maximum image aspect ratio is 20:1.

Testing

- Maximum number of testing datasets in a version of a model is 1.

- Maximum dataset manifest file size is 1 GB.
- Minimum number of unique labels per Objects, Scenes, and Concepts (classification) dataset is 2.
- Minimum number of unique labels per Object Location (detection) dataset is 1.
- Maximum number of unique labels per dataset is 250.
- Minimum number of images per label is 0.
- Maximum number of images per label is 1000.
- Maximum number of images per Object Location (detection) dataset is 250,000.

The limit for *Asia Pacific (Mumbai)* and *Europe (London)* AWS Regions is 7,000 images.

- Maximum number of images per Objects, Scenes, and Concepts (classification) dataset is 500,000. The default is 250,000. To request an increase, see [Create Case](#).

The limit for *Asia Pacific (Mumbai)* and *Europe (London)* AWS Regions is 7,000 images. You can't request a limit increase.

- Minimum number of labels per image per manifest is 0.
- Maximum number of labels per image per manifest is 50.
- Minimum number of bounding boxes in an image per manifest is 0.
- Maximum number of bounding boxes in an image per manifest is 50.
- Minimum image dimension of an image file in an Amazon S3 bucket is 64 pixels x 64 pixels.
- Maximum image dimension of an image file in an Amazon S3 bucket is 4096 pixels x 4096 pixels.
- Maximum file size for an image in an Amazon S3 bucket is 15 MB.
- Supported file formats are PNG and JPEG image formats.
- Maximum image aspect ratio is 20:1.

Detection

- Maximum size of images passed as raw bytes is 4 MB.
- Maximum file size for an image in an Amazon S3 bucket is 15 MB.
- Minimum image dimension of an input image file (stored in an Amazon S3 bucket or supplied as image bytes) is 64 pixels x 64 pixels.
- Maximum image dimension of an input image file (stored in an Amazon S3 or supplied as image bytes) is 4096 pixels x 4096 pixels.
- Supported file formats are PNG and JPEG image formats.
- Maximum image aspect ratio is 20:1.

Model copying

- The maximum number of project policies that you can [attach \(p. 287\)](#) to a project is 5.
- The maximum number of concurrent copy jobs in a destination is 5.

Amazon Rekognition Custom Labels API reference

The Amazon Rekognition Custom Labels API is documented as part of the Amazon Rekognition API reference content. This is a list of the Amazon Rekognition Custom Labels API operations with links to the appropriate Amazon Rekognition API reference topic. Also, API reference links within this document go to the appropriate Amazon Rekognition Developer Guide API reference topic. For information about using the API, see

[This section gives you an overview of the workflow to train and use an Amazon Rekognition Custom Labels model with the console and the AWS SDK.](#)

Note

Amazon Rekognition Custom Labels now manages datasets within a project. You can create datasets for your projects with the console and with the AWS SDK. If you have previously used Amazon Rekognition Custom Labels, your older datasets might need associating with a new project. For more information, see [\(Optional\) Step 7: Associate prior datasets with new projects \(p. 10\)](#)

Topics

- [Decide your model type \(p. 27\)](#)
- [Create a model \(p. 29\)](#)
- [Improve your model \(p. 31\)](#)
- [Start your model \(p. 32\)](#)
- [Analyze an image \(p. 32\)](#)
- [Stop your model \(p. 33\)](#)

Decide your model type

You first decide which type of model you want to train, which depends on your business goals. For example, you could train a model to find your logo in social media posts, identify your products on store shelves, or classify machine parts in an assembly line.

Amazon Rekognition Custom Labels can train the following types of model:

- [Find objects, scenes, and concepts \(p. 27\)](#)
- [Find object locations \(p. 28\)](#)
- [Find the location of brands \(p. 29\)](#)

To help you decide which type of model to train, Amazon Rekognition Custom Labels provides example projects that you can use. For more information, see [Getting started with Amazon Rekognition Custom Labels \(p. 11\)](#).

Find objects, scenes, and concepts

The model predicts objects, scenes, and concepts associated with an entire image. For example, you can train a model that determines if an image contains a *tourist attraction*, or not.

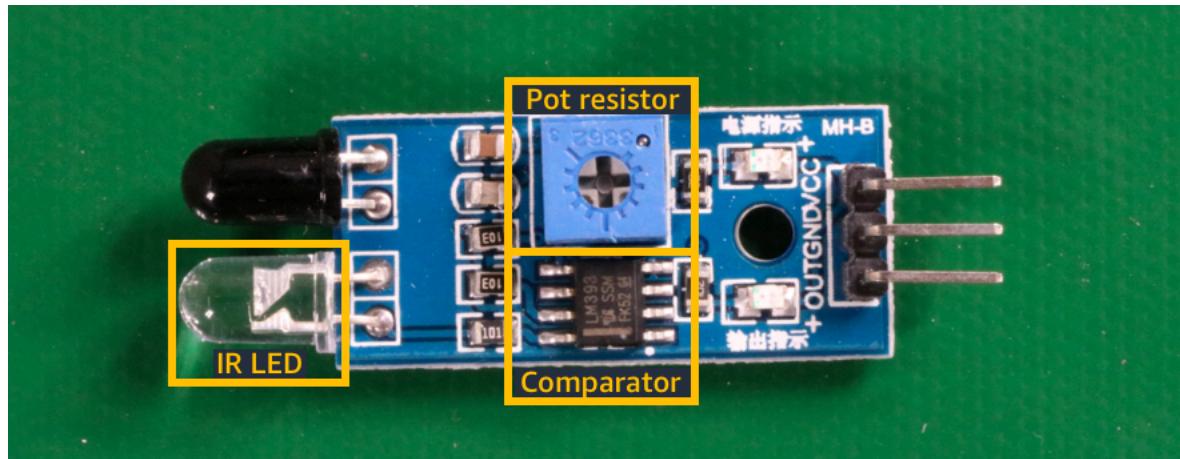


Alternatively, you can train a model that categorizes images into multiple categories. For example, the previous image might have categories such as *sky color*, *reflection*, or *lake*.

The [Image classification \(p. 12\)](#) and [Multi-label image classification \(p. 12\)](#) example projects show different uses for image-level labels.

Find object locations

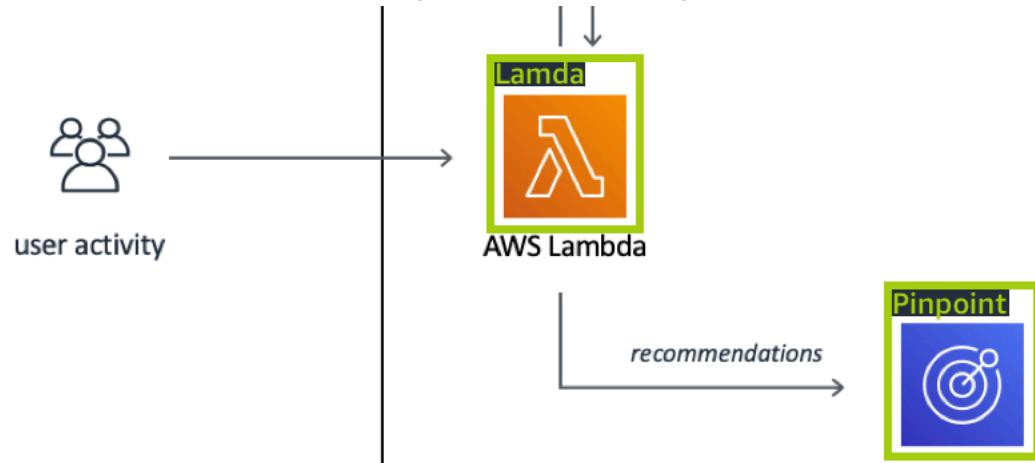
The model predicts the location of an object on an image. The prediction includes bounding box information for the object location and a label that identifies the object within the bounding box. For example, the following image shows bounding boxes around various parts of a circuit board, such as a *comparator* or *pot resistor*.



The Object localization (p. 13) example project shows how Amazon Rekognition Custom Labels uses labeled bounding boxes to train a model that finds object locations.

Find the location of brands

Amazon Rekognition Custom Labels can train a model that finds the location of brands, such as logos, on an image. The prediction includes bounding box information for the brand location and a label that identifies the object within the bounding box.



Create a model

The steps to create a model are creating a project, creating training and test datasets, and training the model.

Create a project

An Amazon Rekognition Custom Labels project is a group of resources needed to create and manage a model. A project manages the following:

- **Datasets** – The images and image labels used to train a model. A project has a training dataset and a test dataset.
- **Models** – The software that you train to find the concepts, scenes, and objects unique to your business. You can have multiple versions of a model in a project.
We recommend that you use a project for a single use case, such as finding circuit board parts on a circuit board.

You can create a project with the Amazon Rekognition Custom Labels console and with the `CreateProject` API. For more information, see [Creating a project \(p. 50\)](#).

Create training and test datasets

A dataset is a set of images and labels that describe those images. Within your project, you create a training dataset and a test dataset that Amazon Rekognition Custom Labels uses to train and test your model.

A label identifies an object, scene, concept, or bounding box around an object in an image. Labels are either assigned to an entire image (*image-level*) or they are assigned to a bounding box that surrounds an object on an image.

Important

How you label the images in your datasets determines the type of model that Amazon Rekognition Custom Labels creates. For example, to train a model that finds objects, scenes and concepts, you assign image level labels to the images in your training and test datasets. For more information, see Purposing datasets (p. 55).

Images must be in PNG and JPEG format, and you should follow the input images recommendations. For more information, see Preparing images (p. 58).

Create training and test datasets (Console)

You can start a project with a single dataset, or with separate training and test datasets. If you start with a single dataset, Amazon Rekognition Custom Labels splits your dataset during training to create a training dataset (80%) and a test dataset (20%) for your project. Start with a single dataset if you want Amazon Rekognition Custom Labels to decide which images are used for training and testing. For complete control over training, testing, and performance tuning, we recommend that you start your project with separate training and test datasets.

To create the datasets for a project, you import the images in one of the following ways:

- Import images from your local computer.
- Import images from an S3 bucket. Amazon Rekognition Custom Labels can label the images using the folder names that contain the images.
- Import an Amazon SageMaker Ground Truth manifest file.
- Copy an existing Amazon Rekognition Custom Labels dataset.

For more information, see [Creating training and test datasets \(Console\) \(p. 59\)](#).

Depending on where you import your images from, your images might be unlabeled. For example, images imported from a local computer aren't labeled. Images imported from an Amazon SageMaker Ground Truth manifest file are labeled. You can use the Amazon Rekognition Custom Labels console to add, change, and assign labels. For more information, see [Labeling images \(p. 99\)](#).

To create your training and test datasets with the console, see [Creating training and test datasets \(Console\) \(p. 59\)](#). For a tutorial that includes creating training and test datasets, see [Tutorial: Classifying images \(p. 34\)](#).

Create training and test datasets (SDK)

To create your training and test datasets, you use the `CreateDataset` API. You can create a dataset by using an Amazon Sagemaker format manifest file or by copying an existing Amazon Rekognition Custom Labels dataset. For more information, see [Create training and test datasets \(SDK\) \(p. 65\)](#). If necessary, you can create your own manifest file. For more information, see the section called "Creating a manifest file" (p. 242).

Train your model

Train your model with the training dataset. A new version of a model is created each time it is trained. During training, Amazon Rekognition Custom Labels test the performance of your trained model. You can use the results to evaluate and improve your model. Training takes a while to complete. You are only charged for a successful model training. For more

information, see [Training an Amazon Rekognition Custom Labels model \(p. 106\)](#). If model training fails, Amazon Rekognition Custom Labels provides debugging information that you can use. For more information, see [Debugging a failed model training \(p. 116\)](#).

Train your model (Console)

To train your model with the console, see [Training a model \(Console\) \(p. 107\)](#).

Training a model (SDK)

You train an Amazon Rekognition Custom Labels model by calling `CreateProjectVersion`.

Improve your model

During testing, Amazon Rekognition Custom Labels creates evaluation metrics that you can use to improve your trained model.

Evaluate your model

Evaluate the performance of your model by using the performance metrics created during testing. Performance metrics, such as F1, precision, and recall, allow you to understand the performance of your trained model, and decide if you're ready to use it in production. For more information, see [Metrics for evaluating your model \(p. 152\)](#).

Evaluate a model (console)

To view performance metrics, see [Accessing evaluation metrics \(Console\) \(p. 155\)](#).

Evaluate a model (SDK)

To get performance metrics, you call `DescribeProjectVersions` to get the testing results. For more information, see [Accessing Amazon Rekognition Custom Labels evaluation metrics \(SDK\) \(p. 156\)](#). The testing results include metrics not available in the console, such as a confusion matrix for classification results. The testing results are returned in the following formats:

- F1 score – A single value representing the overall performance of precision and recall for the model. For more information, see [F1 \(p. 154\)](#).
- Summary file location – The testing summary includes aggregated evaluation metrics for the entire testing dataset and metrics for each individual label. `DescribeProjectVersions` returns the S3 bucket and folder location of the summary file. For more information, see [Summary file \(p. 157\)](#).
- Evaluation manifest snapshot location – The snapshot contains details about the test results, including the confidence ratings and the results of binary classification tests, such as false positives. `DescribeProjectVersions` returns the S3 bucket and folder location of the snapshot files. For more information, see [Evaluation manifest snapshot \(p. 158\)](#).

Improve your model

If improvements are needed, you can add more training images or improve dataset labeling. For more information, see [Improving an Amazon Rekognition Custom Labels model \(p. 163\)](#). You can also give feedback on the predictions your model makes and use it to make improvements to your model. For more information, see [Model feedback solution \(p. 309\)](#).

Improve your model (console)

To add images to a dataset, see [Adding more images to a dataset \(p. 218\)](#). To add or change labels, see the section called “Labeling images” (p. 99).

To retrain your model, see [Training a model \(Console\) \(p. 107\)](#).

Improve your model (SDK)

To add images to a dataset or change the labeling for an image, use the `UpdateDatasetEntries` API. `UpdateDatasetEntries` updates or adds JSON lines to a manifest file. Each JSON line contains information for a single image, such as assigned labels or bounding box information. For more information, see [Adding more images \(SDK\) \(p. 218\)](#). To view the entries in a dataset, use the `ListDatasetEntries` API.

To retrain your model, see [Training a model \(SDK\) \(p. 110\)](#).

Start your model

Before you can use your model, you start the model by using the Amazon Rekognition Custom Labels console or the `StartProjectVersion` API. You are charged for the amount of time that your model runs. For more information, see [Running a trained model \(p. 165\)](#).

Start your model (console)

To start your model using the console, see [Starting an Amazon Rekognition Custom Labels model \(Console\) \(p. 168\)](#).

Start your model

You start your model calling `StartProjectVersion`. For more information, see [Starting an Amazon Rekognition Custom Labels model \(SDK\) \(p. 168\)](#).

Analyze an image

To analyze an image with your model, you use the `DetectCustomLabels` API. You can specify a local image, or an image stored in an S3 bucket. The operation also requires the Amazon Resource Name (ARN) of the model that you want to use.

If your model finds objects, scenes, and concepts, the response includes a list of image-level labels found in the image. For example, the following image shows the image-level labels found using *Rooms* example project.



If the model finds object locations, the response includes list of labeled bounding boxes found in the image. A bounding box represents the location of an object on an image. You can use the bounding box information to draw a bounding box around an object. For example, the following image shows bounding boxes around circuit board parts found using the *Circuit boards* example project.



For more information, see [Analyzing an image with a trained model \(p. 182\)](#).

Stop your model

You are charged for the time that your model is running. If you are no longer using your model, stop the model by using the Amazon Rekognition Custom Labels console, or by using the `StopProjectVersion` API. For more information, see [Stopping an Amazon Rekognition Custom Labels model \(p. 175\)](#).

Stop your model (SDK)

To stop a running model with the console, see [Stopping an Amazon Rekognition Custom Labels model \(Console\) \(p. 175\)](#).

Stop your model (SDK)

To stop a running model, call `StopProjectVersion`. For more information, see [Stopping an Amazon Rekognition Custom Labels model \(SDK\) \(p. 176\)](#).

(p. 27).

Training your model

Projects

- [CreateProject](#) — Creates your Amazon Rekognition Custom Labels project which is a logical grouping of resources (images, Labels, models) and operations (training, evaluation, and detection).
- [DeleteProject](#) — Deletes an Amazon Rekognition Custom Labels project.
- [DescribeProjects](#) — Returns a list of all your Amazon Rekognition Custom Labels projects.

Project Policies

- [PutProjectPolicy](#) — Attaches a project policy to a Amazon Rekognition Custom Labels project in a trusting AWS account.
- [ListProjectPolicies](#) — Returns a list of the project policies attached to a project.
- [DeleteProjectPolicy](#) — Deletes an existing project policy.

Datasets

- [CreateDataset](#) — Creates a Amazon Rekognition Custom Labels dataset.
- [DeleteDataset](#) — Deletes an Amazon Rekognition Custom Labels dataset.
- [DescribeDataset](#) — Describes an Amazon Rekognition Custom Labels dataset.
- [DistributeDatasetEntries](#) — Distributes the entries (images) in a training dataset across the training dataset and the test dataset for a project.
- [ListDatasetEntries](#) — Returns a list of entries (images) in an Amazon Rekognition Custom Labels dataset.
- [ListDatasetLabels](#) — Returns a list of labels assigned to an Amazon Rekognition Custom Labels dataset.
- [UpdateDatasetEntries](#) — Adds or updates entries (images) in an Amazon Rekognition Custom Labels dataset.

Models

- [CreateProjectVersion](#) — Trains your Amazon Rekognition Custom Labels model.
- [CopyProjectVersion](#) — Copies your Amazon Rekognition Custom Labels model.
- [DeleteProjectVersion](#) — Deletes an Amazon Rekognition Custom Labels model.
- [DescribeProjectVersions](#) — Returns a list of all the Amazon Rekognition Custom Labels models within a specific project.

Tags

- [TagResource](#) — Adds one or more key-value tags to an Amazon Rekognition Custom Labels model.
- [UntagResource](#) — Removes one or more tags from an Amazon Rekognition Custom Labels model.

Using your model

- [DetectCustomLabels](#) — Analyzes an image with your custom labels model.
- [StartProjectVersion](#) — Starts your custom labels model.
- [StopProjectVersion](#) — Stops your custom labels model.

Document history for Amazon Rekognition Custom Labels

The following table describes important changes in each release of the *Amazon Rekognition Custom Labels Developer Guide*. For notification about updates to this documentation, you can subscribe to an RSS feed.

- **Latest documentation update:** August 16th, 2022

Change	Description	Date
Amazon Rekognition Custom Labels can now copy trained models (p. 333)	You can now copy a trained model from one AWS account to another AWS account within the same AWS Region. For more information, see Copying an Amazon Rekognition Custom Labels model (SDK) .	August 16, 2022
Amazon Rekognition Custom Labels can now automatically scale inference units. (p. 333)	To help with spikes in demand, Amazon Rekognition Custom Labels can now scale the number of inference units that your model uses. For more information, see Running a trained Amazon Rekognition Custom Labels model .	August 16, 2022
Create a manifest file from a CSV file (p. 333)	You can now simplify the creation of a manifest file by using a script that reads image classification information from a CSV file. For more information, see Creating a manifest file from a CSV file .	February 2, 2022
Amazon Rekognition Custom Labels now manages datasets with projects (p. 333)	You can use projects to manage the training and test datasets that you use to create a model. For more information, see Understanding Amazon Rekognition Custom Labels .	November 1, 2021
Amazon Rekognition Custom Labels is integrated with AWS CloudFormation (p. 333)	You can use AWS CloudFormation to provision and configure Amazon Rekognition Custom Labels projects. For more information, see Creating a project with AWS CloudFormation .	October 21, 2021

Updated getting started experience (p. 333)	The Amazon Rekognition Custom Labels console now includes tutorial videos and example projects. For more information, see Getting started with Amazon Rekognition Custom Labels .	July 22, 2021
Updated information about thresholds and using metrics (p. 333)	Information about setting a desired threshold value by using the <code>MinConfidence</code> input parameter to <code>DetectCustomLabels</code> . For more information, see Analyzing an image with a trained model .	June 8, 2021
Added AWS KMS key support (p. 333)	You can now use your own KMS key to encrypt your training and test images. For more information, see Training a model .	May 19, 2021
Added tagging (p. 333)	Amazon Rekognition Custom Labels now supports tagging. You can use tags to identify, organize, search for, and filter your Amazon Rekognition Custom Labels models. For more information, see Tagging a model .	March 25, 2021
Updated setup topic (p. 333)	Updated setup information on how to encrypt training files. For more information, see Step 5: (Optional) Encrypting training files .	March 18, 2021
Added dataset copy topic (p. 333)	Information on how to copy a dataset to a different AWS Region. For more information, see Copying a dataset to a different AWS region .	March 5, 2021
Added Amazon SageMaker GroundTruth multi-label manifest transform topic (p. 333)	Information on how to transform an Amazon SageMaker GroundTruth multi-label format manifest to a Amazon Rekognition Custom Labels format manifest file. For more information, see Transforming multi-label SageMaker Ground Truth manifest files .	February 22, 2021

Added debugging information for model training (p. 333)	You can now use validation results manifests to get in-depth debugging information about model training errors. For more information, see Debugging a failed model training .	October 8, 2020
Added COCO transform information and example (p. 333)	Information on how to transform a COCO object detection format dataset into an Amazon Rekognition Custom Labels manifest file. For more information, see Transforming COCO datasets .	September 2, 2020
Amazon Rekognition Custom Labels now supports single object training (p. 333)	To create an Amazon Rekognition Custom Labels model that finds the location of a single object, you can now create a dataset that only requires one label. For more information, see Drawing bounding boxes .	June 25, 2020
Project and model delete operations added (p. 333)	You can now delete Amazon Rekognition Custom Labels projects and models with the console and with the API. For more information, see Deleting an Amazon Rekognition Custom Labels Model and Deleting an Amazon Rekognition Custom Labels project	April 1, 2020
Added Java examples (p. 333)	Added Java examples covering project creation, model training, model running, and image analysis.	December 13, 2019
New feature and guide (p. 333)	This is the initial release of the Amazon Rekognition Custom Labels feature and the <i>Amazon Rekognition Custom Labels Developer Guide</i> .	December 3, 2019

AWS glossary

For the latest AWS terminology, see the [AWS glossary](#) in the *AWS General Reference*.